

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

M269 Units 3,4,5

Sorting, Searching, Graph Algorithms

Phil Molyneux

13 March 2021

M269 Units 3,4,5

Agenda & Aims

- ▶ Welcome and introductions
- ▶ Session on M269 *Sorting, Abstract Data Types, Searching, Graph Algorithms*
- ▶ Recursion
- ▶ String search
- ▶ Bags and Abstract Data Types (and Objects)
- ▶ Search Trees and Height Balanced Trees
- ▶ Graph Algorithms, Greedy algorithm example — Interval Scheduling
- ▶ Dynamic Programming — Edit Distance example
- ▶ Implementations in Structured English, Python and Haskell (Optional)
- ▶ *Note* there is more material here than we can cover — some is for optional interest

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

M269 Units 3,4,5

Slides, Notes, Files

- ▶ Slides, notes and source code files are at
- ▶ [M269 Tutorial Sorting](#)
- ▶ [M269 Tutorial Binary Trees](#)
- ▶ [M269 Tutorial Graphs](#)
- ▶ [M269 Tutorial Units 3,4,5](#)

M269 Units 3,4,5

Phil Molyneux

Agenda

[Adobe Connect](#)

[Recursion](#)

[Sorting](#)

[Searching](#)

[String Search](#)

[Bags](#)

[Abstract Data Types](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic
Programming](#)

[Future Work](#)

[Web Sites &
References](#)

M269 Tutorial

Introductions — Me

- ▶ *Name* Phil Molyneux
- ▶ *Background* Physics & Maths, Operational Research, Computer Science
- ▶ *First programming languages* Fortran, BASIC, Pascal
- ▶ *Favourite Software*
 - ▶ Haskell — pure functional programming language
 - ▶ Text editors TextMate, Sublime Text — previously Emacs
 - ▶ Word processing in L^AT_EX — all these slides and notes
 - ▶ Mac OS X
- ▶ *Learning style* — I read the manual before using the software

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

M269 Tutorial

Introductions — You

- ▶ *Name ?*
- ▶ *New topics last month ?*
- ▶ *M269 Unit 3,4,5 Sorting, Searching, Graph Algorithm topics you want covered ?*
- ▶ *Learning style ?*
- ▶ *Other OU courses ?*
- ▶ *Anything else ?*

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Adobe Connect

Interface — Student Quick Reference

Participant Quick Reference Guide

The screenshot shows the Adobe Connect interface with the following labeled components:

- Share pod:** Located at the top left, containing icons for Meeting, Audio set up, Speaker volume, Webcam, and a Share button.
- Connect microphone:** A button located below the Share pod.
- Status:** A text box below the Connect microphone button containing the text: "Status: raise hand, agree, disagree, step away, speak louder, speak softer, speed up, slow down, laughter, applause".
- Nothing is being shared:** A message displayed in the main content area.
- Adobe Connect Help:** A button located at the top right.
- Connection status:** A button located below the Adobe Connect Help button.
- Video pod:** A large area on the right side of the interface, currently displaying a "Start My Webcam" button.
- Attendee pod:** A list of participants located below the Video pod, showing "Attendees (1)", "User Openen 2", "Hosts (0)", "Presenters (0)", and "Participants (1)".
- Chat pod:** A chat window located at the bottom right, showing "Chat (Everyone)" and a text input field.

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

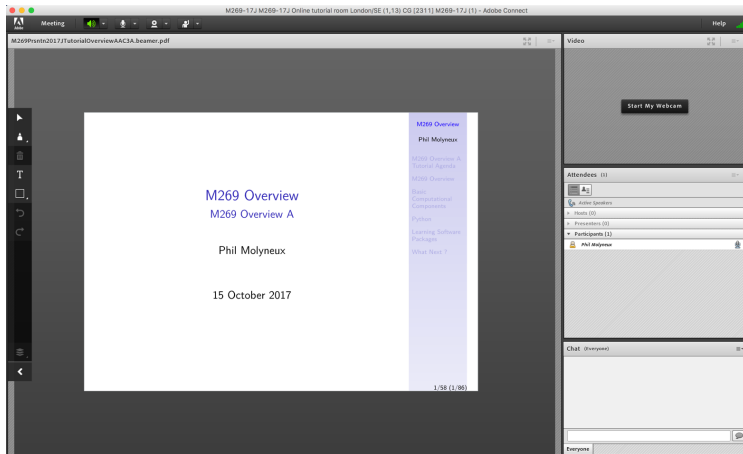
Dynamic Programming

Future Work

Web Sites & References

Adobe Connect

Interface — Student View



M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Adobe Connect

Settings

- ▶ **Everybody: Audio Settings** Meeting > Audio Setup Wizard. . .
- ▶ **Audio** Menu bar > Audio > Microphone rights for Participants ✓
- ▶ Do not *Enable single speaker mode*
- ▶ **Drawing Tools** Share pod menu bar > Draw (1 slide/screen)
- ▶ Share pod menu bar > Menu icon > Enable Participants to draw ✓ gray
- ▶ Meeting > Preferences > Whiteboard > Enable Participants to draw ✓
- ▶ Cancel hand tool . . . Do not enable green pointer. . .
- ▶ Meeting > Preferences > Attendees Pod ✗ *Raise Hand notification*
- ▶ Meeting > Preferences > Display Name Display First & Last Name
- ▶ **Cursor** Meeting > Preferences > General tab > Host Cursors
Show to all attendees ✓ (default *Off*)
- ▶ Meeting > Preferences > Screen Share > Cursor > Show Application Cursor
- ▶ **Webcam** Menu bar > Webcam > Enable Webcam for Participants ✓
- ▶ **Recording** Meeting > Record Meeting. . . ✓

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Adobe Connect

Access

► Tutor Access

TutorHome > M269 Website > Tutorials

Cluster Tutorials > M269 Online tutorial room

Tutor Groups > M269 Online tutor group room

Module-wide Tutorials > M269 Online module-wide room

► Attendance

TutorHome > Students > View your tutorial timetables

► Beamer Slide Scaling 440% (422 x 563 mm)

► Clear Everyone's Status

Attendee Pod > Menu > Clear Everyone's Status

► Grant Access and send link via email

Meeting > Manage Access & Entry > Invite Participants...

► Presenter Only Area

Meeting > Enable/Disable Presenter Only Area

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Adobe Connect

Keystroke Shortcuts

- ▶ Keyboard shortcuts in Adobe Connect
- ▶ **Toggle Mic**  + **M** (Mac), **Ctrl** + **M** (Win) (On/Disconnect)
- ▶ **Toggle Raise-Hand status**  + **E**
- ▶ **Close dialog box**  (Mac), **Esc** (Win)
- ▶ **End meeting**  + ****

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

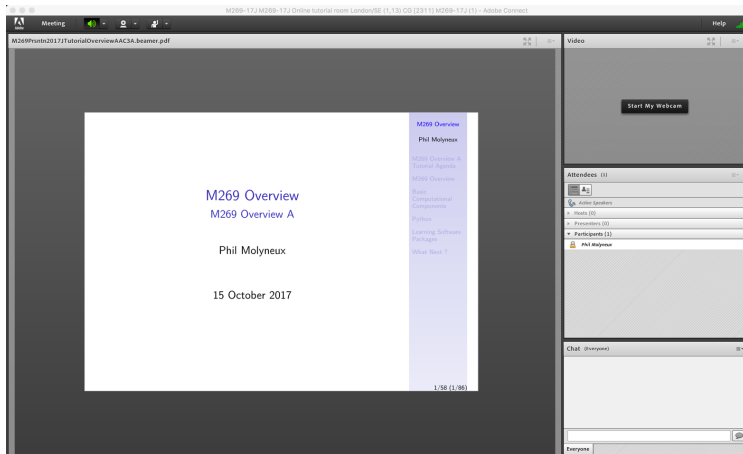
Dynamic Programming

Future Work

Web Sites & References

Adobe Connect Interface

Student View (default)



M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

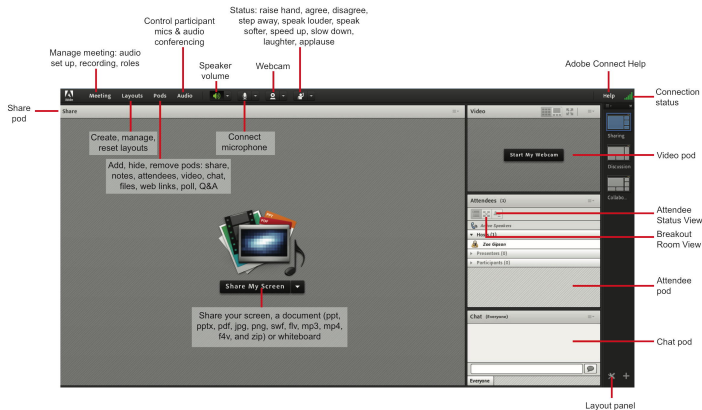
Future Work

Web Sites & References

Adobe Connect Interface

Tutor View

Host Quick Reference Guide



M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

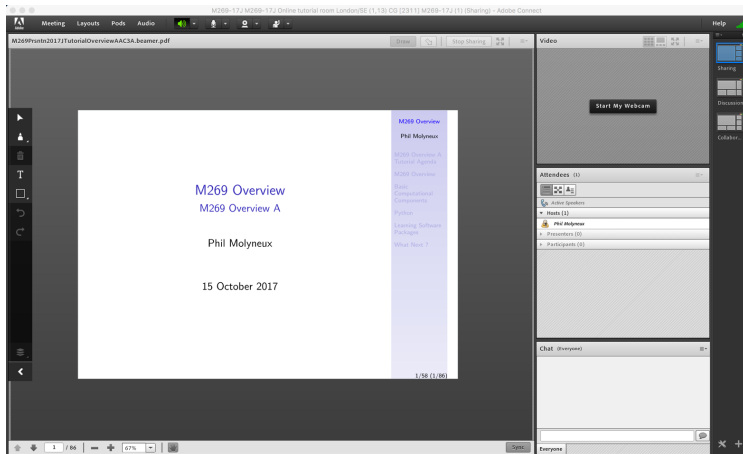
Dynamic Programming

Future Work

Web Sites & References

Adobe Connect Interface

Tutor View



M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Adobe Connect Interface

Sharing Screen & Applications

- ▶ **Share My Screen** > **Application tab** > **Terminal** for **Terminal**
- ▶ **Share menu** > **Change View** > **Zoom in** for mismatch of screen size/resolution (Participants)
- ▶ (Presenter) Change to 75% and back to 100% (solves participants with smaller screen image overlap)
- ▶ Leave the application on the original display
- ▶ Beware blue hatched rectangles — from other (hidden) windows or contextual menus
- ▶ Presenter screen pointer affects viewer display — beware of moving the pointer away from the application
- ▶ First time: **System Preferences** > **Security & Privacy** > **Privacy** > **Accessibility**

Agenda

Adobe Connect

Student View
Settings
Student & Tutor Views
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Adobe Connect

Ending a Meeting

- ▶ *Notes for the tutor only*
- ▶ **Student:** Meeting > Exit Adobe Connect
- ▶ **Tutor:**
- ▶ **Recording** Meeting > Stop Recording ✓
- ▶ **Remove Participants** Meeting > End Meeting... ✓
 - ▶ Dialog box allows for message with default message:
 - ▶ *The host has ended this meeting. Thank you for attending.*
- ▶ **Recording availability** *In course Web site for joining the room, click on the eye icon in the list of recordings under your recording* — edit description and name
- ▶ **Meeting Information** Meeting > Manage Meeting Information — can access a range of information in Web page.
- ▶ **Attendance Report** see course Web site for joining room

Agenda

Adobe Connect

Student View
Settings
Student & Tutor Views
Sharing Screen & Applications

Ending a Meeting

Invite Attendees
Layouts
Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Adobe Connect

Invite Attendees

- ▶ **Provide Meeting URL** Menu Meeting Manage Access & Entry
Invite Participants...
- ▶ **Allow Access without Dialog** Menu Meeting
Manage Meeting Information provides new browser window with Meeting Information Tab bar Edit Information
- ▶ Check *Anyone who has the URL for the meeting can enter the room*
- ▶ Default *Only registered users and accepted guests may enter the room*
- ▶ **Reverts to default next session but URL is fixed**
- ▶ Guests have blue icon top, registered participants have yellow icon top — same icon if URL is open
- ▶ See [Start, attend, and manage Adobe Connect meetings and sessions](#)

Agenda

Adobe Connect

Student View
Settings
Student & Tutor Views
Sharing Screen & Applications
Ending a Meeting
Invite Attendees

Layouts
Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

- ▶ **Creating new layouts** example *Sharing* layout
- ▶ **Menu** ▶ **Layouts** ▶ **Create New Layout...** ▶ **Create a New Layout dialog**
▶ **Create a new blank layout** and name it *PMolyMain*
- ▶ New layout has no Pods but does have Layouts Bar open (see Layouts menu)
- ▶ **Pods**
- ▶ **Menu** ▶ **Pods** ▶ **Share** ▶ **Add New Share** and resize/position — initial name is *Share n*
- ▶ **Rename Pod** **Menu** ▶ **Pods** ▶ **Manage Pods...** ▶ **Manage Pods**
▶ **Select** ▶ **Rename** or **Double-click & rename**
- ▶ Add Video pod and resize/reposition
- ▶ Add Attendance pod and resize/reposition
- ▶ Add Chat pod — name it *PMolyChat* — and resize/reposition

Agenda

Adobe Connect

Student View
Settings
Student & Tutor Views
Sharing Screen & Applications
Ending a Meeting
Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Adobe Connect

Layouts

- ▶ Dimensions of **Sharing** layout (on 27-inch iMac)
 - ▶ Width of Video, Attendees, Chat column 14 cm
 - ▶ Height of Video pod 9 cm
 - ▶ Height of Attendees pod 12 cm
 - ▶ Height of Chat pod 8 cm
- ▶ **Duplicating Layouts** does *not* give new instances of the Pods and is probably not a good idea (apart from local use to avoid delay in reloading Pods)

Agenda

Adobe Connect

Student View
Settings
Student & Tutor Views
Sharing Screen & Applications
Ending a Meeting
Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Adobe Connect

Chat Pods

- ▶ **Format Chat text**

- ▶  menu icon 

- ▶ Choices: Red, Orange, Green, Brown, Purple, Pink, Blue, Black

- ▶ Note: Color reverts to Black if you switch layouts

- ▶  menu icon 

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Recursion Examples

Introduction

- ▶ These notes give several examples of recursive programs similar to the Python Activities in M269 Unit 3
- ▶ Each example gives a function definition for the stated problem followed by a sample evaluation of the function applied to a small argument
- ▶ This is followed by a version of the program code similar to that of Python Activities 3.2 to 3.5 with print statements giving the value of some variables on call and return of the function
- ▶ Since print statements embedded in a recursive function can be as hard to follow as the recursion itself, there are two further implementations with pure functions that accumulate the call and return messages and also a version which returns the evaluation messages for a sample evaluation — this aims to give a further perspective on thinking about recursion

Agenda

Adobe Connect

Recursion

Recursion Examples — Introduction

Factorial

String Prefixes Example

Recursion Examples — Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Recursion Example 1

Factorial

- ▶ `factorial()` is a standard introduction to recursion
- ▶ $\text{factorial}(n) = n * (n-1) * \dots * 2 * 1$

```
10 def factorial(n) :  
11     if (n == 1) :           # line (A)  
12         return 1  
13     else :                  # line (B)  
14         return (n * factorial(n - 1))
```

- ▶ Below is a sample evaluation of `factorial(6)`
- ▶ To evaluate a function applied to actual arguments, we evaluate the body of the function with each formal parameter replaced by the corresponding actual argument
- ▶ In the evaluation below we refer to line (A) or 11 and line (B) or 13 (of file `Recursion2020PrintCallReturnEval.py`)

Factorial (2)

Sample Evaluation

```
# factorial(6) evaluation
Call 1 -> 6 * factorial(5)
           # by line (B)
Call 2 -> 6 * (5 * factorial(4))
           # by line (B)
Call 3 -> 6 * (5 * (4 * factorial(3)))
           # by line (B)
Call 4 -> 6 * (5 * (4 * (3 * factorial(2))))
           # by line (B)
Call 5 -> 6 * (5 * (4 * (3 * (2 * factorial(1)))))
           # by line (B)
Call 6 -> 6 * (5 * (4 * (3 * (2 * (1 * 1)))))
           # by line (A)
factorial(6) = 720
```

- ▶ In writing down an evaluation by hand it is important to hang on to the brackets — they indicate a subexpression that has to be evaluated before an outer expression
- ▶ It is possible to write the definition differently to accumulate the product as the computation goes — see later

Agenda

Adobe Connect

Recursion

Recursion Examples —
Introduction
Factorial

Factorial Definition

Factorial with Print
Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (3)

Factorial with embedded print statements

```

18 def facWithPrint(n) :
19     return helpFacWithPrint(n, 1, 1)

21 def helpFacWithPrint(n, call, prodAccum) :
22     if (n == 1) :                # line (A)
23         callMsg = calcFacCallMsg(call, n)
24         print(callMsg)
25         retMsg = calcFacRetMsg(call, n, prodAccum)
26         print(retMsg)
27         return (n * prodAccum)
28     else :                       # line (B)
29         callMsg = calcFacCallMsg(call, n)
30         print(callMsg)
31         pAccum = helpFacWithPrint(n-1, call+1
32                                 , prodAccum)
33         retMsg = calcFacRetMsg(call, n, pAccum)
34         print(retMsg)
35         return (n * pAccum)

```

- Below is the output from `facWithPrint(6)` and some comments on the code

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print
Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (4)

facWithPrint(6) output

```
Python3>>> facWithPrint(6)
Call 1 of function with n = 6
Call 2 of function with n = 5
Call 3 of function with n = 4
Call 4 of function with n = 3
Call 5 of function with n = 2
Call 6 of function with n = 1
Return call 6 returning 1 * 1
Return call 5 returning 2 * 1
Return call 4 returning 3 * 2
Return call 3 returning 4 * 6
Return call 2 returning 5 * 24
Return call 1 returning 6 * 120
720
Python3>>>
```

- ▶ Here Python3>>> is my Python shell prompt
- ▶ Below are some comments on the code

Agenda

Adobe Connect

Recursion

Recursion Examples —
Introduction

Factorial

Factorial Definition

Factorial with Print
Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (5)

Factorial with embedded print statements — description

- ▶ `facWithPrint()`, `helpFacWithPrint()` description
- ▶ `facWithPrint()` calls `helpFacWithPrint(n, 1, 1)`
- ▶ The arguments to `helpFacWithPrint()` are:
- ▶ `n` initialised to the argument of `facWithPrint()`
- ▶ `call` the call number initialised to 1
- ▶ `prodAccum` is the base case (initialised to 1) — this argument is not needed but there may be generalisations
- ▶ The recursive call to `helpFacWithPrint()` at line 31 increments `call` — this mimics a global variable
- ▶ The calls to `calcFacCallMsg()`, `calcFacRetMsg()` at lines 23,29,25,33 construct the messages which are then printed — we could have lifted the two calls to `calcFacCallMsg` to just one before the `if`

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print Statements

Factorial with Call, Return Messages

Factorial with Evaluation Messages

String Prefixes Example

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (6)

Factorial with Call, Return Messages

- ▶ The snag with inserting print statements into code is the program ceases to be a pure function — rather it is a program that takes a string and returns a computation which does some I/O and returns the factorial
- ▶ This conflates understanding the original pure function with understanding where the output comes from in code with side effects
- ▶ This can be written as a pure function which takes a number and returns a triple of the output number, and lists of call and return messages, and then decide how to present this

```
41 def funcFacWithMsgs(n) :  
42   return helpFuncFacWM(n, 1, 1, [], [])
```

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print
Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (7)

helpFuncFacWM()

```
44 def helpFuncFacWM(n, call, prodAccum
45                    , callMsgList, retMsgList) :
46     callMsg = calcFacCallMsg(call, n)
47     if (n == 1) :                               # line (A)
48         retMsg = calcFacRetMsg(call, n, prodAccum)
49         return (n * prodAccum
50                , [callMsg] + callMsgList
51                , [retMsg] + retMsgList )
52     else :                                       # line (B)
53         (pAccum, pCallMsgList, pRetMsgList) \
54         = helpFuncFacWM(n-1, call+1, prodAccum
55                        , callMsgList, retMsgList)
56         retMsg = calcFacRetMsg(call, n, pAccum)
57         return (n * pAccum
58                , [callMsg] + pCallMsgList
59                , [retMsg] + pRetMsgList )
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print

Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References 27/2002

Factorial (7a)

helpFuncFacWM() Description

- ▶ `helpFuncFacWM()` description — the arguments are:
- ▶ `n` initialised to argument of `funcFacWithMsgs()`
- ▶ `call` the call number initialised to 1
- ▶ `prodAccum` is the base case (initialised to 1) — this argument is not needed but there may be generalisations
- ▶ `callMsgList`, the call messages list, initialised to an empty list
- ▶ `retMsgList`, the return messages list, initialised to an empty list
- ▶ The recursive call to `helpFuncFacWM()` at line 54 increments `call` — mimics a global variable
- ▶ The call to `calcFacCallMsg()` is now before the `if` at line 46 — constructs the call message
- ▶ The calls to `calcFacRetMsg()` at lines 48,56 construct the return message

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print

Statements

Factorial with Call, Return Messages

Factorial with Evaluation Messages

String Prefixes Example

Recursion Examples — Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (8)

calcFacCallMsg(), calcFacRetMsg()

```
61 def calcFacCallMsg(call, n) :  
62     return ("Call_" + str(call)  
63            + "_of_function_with_n_" + str(n) )  
  
65 def calcFacRetMsg(call, n, acc) :  
66     return ("Return_call_" + str(call)  
67            + "_returning_" + str(n)  
68            + "_*__" + str(acc) )
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Recursion Examples —
Introduction

Factorial

Factorial Definition

Factorial with Print
Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References 29/202

Factorial (9)

Utility Code for Formatting Output

```
72 nl = "\n"
74 sp = "  "
75 sp20 = sp.join(['']*21) # string of 20 spaces

77 def pStr(aStr) :
78     return ("\" + aStr + "\")

80 def printFacWM(n) :
81     (facFinal, callMsgList, retMsgList) \
82     = funcFacWithMsgs(n)
83     print(nl.join(callMsgList)
84           + nl
85           + nl.join(reversed(retMsgList))
86           + nl + "factorial("
87           + str(n) + ")="
88           + str(facFinal) )
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print

Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References 30/202

Factorial (9a)

Utility Code for Formatting Output — Description

- ▶ `n1`, `sp` are defined to make the code easier to read
- ▶ `join()` and `reversed()` are Python built-in functions
- ▶ The return messages list is reversed to get the messages in the same order as the print statement version — this might take some thinking
- ▶ `pStr()` is needed since `str("")` does not produce `''`

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print

Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (10)

Factorial with Call, Return Messages — Output

► Sample output from `printFacWM(6)`

```
Python3>>> printFacWM(6)
Call 1 of function with n = 6
Call 2 of function with n = 5
Call 3 of function with n = 4
Call 4 of function with n = 3
Call 5 of function with n = 2
Call 6 of function with n = 1
Return call 6 returning 1 * 1
Return call 5 returning 2 * 1
Return call 4 returning 3 * 2
Return call 3 returning 4 * 6
Return call 2 returning 5 * 24
Return call 1 returning 6 * 120
factorial(6) = 720
Python3>>>
```

Agenda

Adobe Connect

Recursion

Recursion Examples — Introduction

Factorial

Factorial Definition

Factorial with Print Statements

Factorial with Call, Return Messages

Factorial with Evaluation Messages

String Prefixes Example

Recursion Examples — Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (11)

Factorial with Evaluation Messages

- ▶ The evaluation messages can also be generated
- ▶ The code requires a bit more thought since the evaluation messages are also recursive

```
92 def funcFacEval(n) :  
93     return helpFuncFacEval(n, 1, 1, [], [])
```

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print
Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (12)

helpFuncFacEval()

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print
Statements

Factorial with Call,
Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References 34/202

```
95 def helpFuncFacEval(n, call, prodAccum
96                        , argList, evalMsgList) :
97     if (n == 1) :                # line (A)
98         reasonStr = "#_by_line_(A)"
99         evalStr = str(n)
100        evalMsg = calcFacEvalMsg(call, n
101                                   , argList, evalStr
102                                   , reasonStr)
103        return (n * prodAccum
104                , [n] + argList, [evalMsg] + evalMsgList )
105    else :                        # line (B)
106        reasonStr = "#_by_line_(B)"
107        evalStr = "factorial(" + str(n-1) + ")"
108        evalMsg = calcFacEvalMsg(call, n
109                                   , argList, evalStr
110                                   , reasonStr)
111        (pAccum, pArgList, pEvalMsgList) \
112        = helpFuncFacEval(n-1, call+1
113                          , prodAccum, [n] + argList, evalMsgList)
114        return (n * pAccum
115                , pArgList, [evalMsg] + pEvalMsgList )
```

Factorial (12a)

helpFuncFacEval() Description

- ▶ `helpFuncFacEval()` description — the arguments are:
- ▶ `n` initialised to the argument of `funcFacEval()`
- ▶ `call` the call number initialised to 1
- ▶ `prodAccum` is the base case (initialised to 1) — this argument is not needed but there may be generalisations
- ▶ `argList`, the previous arguments list, initialised to an empty list — the previous arguments are needed to calculate the evaluation message
- ▶ `evalMsgList`, the evaluation messages list, initialised to an empty list
- ▶ The recursive call to `helpFuncFacEval()` at line 112 increments `call` — this mimics a global variable
- ▶ The calls to `calcFacEvalMsg()` at lines 100,108 construct the evaluation message (see description below)

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print Statements

Factorial with Call, Return Messages

Factorial with Evaluation Messages

String Prefixes Example

Recursion Examples — Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (13)

calcFacEvalMsg()

```
117 def calcFacEvalMsg(call, n
118                        , argList, evalStr
119                        , reasonStr) :
120     if (argList == []) :
121         retMsg = ("Call_" + str(call)
122                  + "_->" + str(n)
123                  + "_*" + evalStr
124                  + nl + sp20 + reasonStr)
125         return retMsg
126     else :
127         nextN = argList[0]
128         nextArgList = argList[1:]
129         nextEvalStr \
130             = "(" + str(n) + "_*_" + evalStr + ")"
131         return calcFacEvalMsg(call, nextN
132                               , nextArgList, nextEvalStr
133                               , reasonStr)
```

- Notice `calcFacEvalMsg()` is recursive unlike `calcFacCallMsg()`, `calcFacRetMsg()`

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print

Statements

Factorial with Call,

Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (13a)

calcFacEvalMsg() Description

- ▶ `calcFacEvalMsg()` description — the arguments are:
- ▶ `call` the current call number
- ▶ `n` the current argument
- ▶ `argList` the list of previous arguments
- ▶ `evalStr` the expression on the right hand side of the current operation (see print output below)
- ▶ `reasonStr` is the string identifying the branch of the `if` used
- ▶ If `argList` is empty, a string is returned
- ▶ Otherwise there is a recursive call to wrap the constructed string with the outer arguments (see print output below)
- ▶ It is easy to get the arguments in the wrong order — perhaps this is why this function is not given often.

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print Statements

Factorial with Call, Return Messages

Factorial with Evaluation Messages

String Prefixes Example

Recursion Examples — Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (14)

Utility Code for Formatting Output

```
145 def printFacEval(n) :  
146     (facFinal, argList, evalMsgList) \  
147     = funcFacEval(n)  
148     print(nl.join(evalMsgList)  
149           + nl + "factorial("  
150           + str(n) + ")_=_"  
151           + str(facFinal) )
```

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print

Statements

Factorial with Call,

Return Messages

Factorial with
Evaluation Messages

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Factorial (15)

Factorial with Evaluation Messages — Output

► Sample output from `printFacEval(6)`

```
Python3>>> printFacEval(6)
Call 1 -> 6 * factorial(5)
                # by line (B)
Call 2 -> 6 * (5 * factorial(4))
                # by line (B)
Call 3 -> 6 * (5 * (4 * factorial(3)))
                # by line (B)
Call 4 -> 6 * (5 * (4 * (3 * factorial(2))))
                # by line (B)
Call 5 -> 6 * (5 * (4 * (3 * (2 * factorial(1)))))
                # by line (B)
Call 6 -> 6 * (5 * (4 * (3 * (2 * (1 * 1)))))
                # by line (A)
factorial(6) = 720
Python3>>>
```

► Looks just like a hand evaluation

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

Factorial Definition

Factorial with Print Statements

Factorial with Call, Return Messages

Factorial with Evaluation Messages

String Prefixes Example

Recursion Examples — Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Recursion Example 2

Concatenating All Prefixes of a String

- ▶ `strPrefs()` takes a string and returns a string which is the concatenation of all prefixes of the string
- ▶ `strPrefs("abc") = "abcaba"`

```

158 def strPrefs(aStr) :
159     if (aStr == "") :           # line (A)
160         return (aStr + "")
161     else :                     # line (B)
162         return (aStr + strPrefs(aStr[:-1]))

```

- ▶ Below is a sample evaluation of `strPrefs"abc"`
- ▶ To evaluate a function applied to actual arguments, we evaluate the body of the function with each formal parameter replaced by the corresponding actual argument
- ▶ In the evaluation below we refer to line (A) or 159 and line (B) or 161 (of file `Recursion2020PrintCallReturnEval.py`)

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes Definition

String Prefixes with Print Statements

String Prefixes with Call, Return Messages

String Prefixes with Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (2)

Sample Evaluation

```
# strPrefs("abc") evaluation
Call 1 -> "abc" + strPrefs("ab")
           # by line (B)
Call 2 -> "abc" + ("ab" + strPrefs("a"))
           # by line (B)
Call 3 -> "abc" + ("ab" + ("a" + strPrefs("")))
           # by line (B)
Call 4 -> "abc" + ("ab" + ("a" + (" " + "")))
           # by line (A)
strPrefs("abc") = "abcaba"
```

- ▶ In writing down an evaluation by hand it is important to hang on to the brackets — they indicate a subexpression that has to be evaluated before an outer expression
- ▶ It is possible to write the definition differently to accumulate the prefixes as the computation goes — see later

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes Definition

String Prefixes with
Print Statements

String Prefixes with
Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (3)

String Prefixes with embedded print statements

```

166 def strPrefsWithPrint(aStr) :
167     return helpStrPrefsWithPrint(aStr,1,"")

169 def helpStrPrefsWithPrint(aStr, call, strAccum) :
170     if (aStr == "") :                # line (A)
171         callMsg = calcStrPrefsCallMsg(call, aStr)
172         print(callMsg)
173         retMsg = calcStrPrefsRetMsg(call, aStr, strAccum)
174         print(retMsg)
175         return (aStr + strAccum)
176     else :                            # line (B)
177         callMsg = calcStrPrefsCallMsg(call, aStr)
178         print(callMsg)
179         pAccum = helpStrPrefsWithPrint(aStr[:-1],call+1
180                                     ,strAccum)
181         retMsg = calcStrPrefsRetMsg(call, aStr, pAccum)
182         print(retMsg)
183         return (aStr + pAccum)

```

- Below is the output from `strPrefsWithPrint("abc")` and some comments on the code

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with
Print Statements

String Prefixes with
Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (4)

strPrefsWithPrint("abc") output

```
Python3>>> strPrefsWithPrint("abc")
Call 1 of function with aStr = "abc"
Call 2 of function with aStr = "ab"
Call 3 of function with aStr = "a"
Call 4 of function with aStr = ""
Return call 4 returning "" + ""
Return call 3 returning "a" + ""
Return call 2 returning "ab" + "a"
Return call 1 returning "abc" + "aba"
'abcaba'
Python3>>>
```

- ▶ Here `Python3>>>` is my Python shell prompt
- ▶ Below are some comments on the code

String Prefixes (5)

String Prefixes with embedded print statements — description

- ▶ `strPrefsWithPrint()`, `helpStrPrefsWithPrint()` description
- ▶ `strPrefsWithPrint()` calls `helpStrPrefsWithPrint(n, 1, 1)`
- ▶ The arguments to `helpStrPrefsWithPrint()` are:
- ▶ `n` initialised to the argument of `strPrefsWithPrint()`
- ▶ `call` the call number initialised to 1
- ▶ `strAccum` is the base case (initialised to "") — this argument is not needed but there may be generalisations
- ▶ The recursive call to `helpStrPrefsWithPrint()` at line 179 increments `call` — mimics a global variable
- ▶ The calls to `calcStrPrefsCallMsg()`, `calcStrPrefsRetMsg()` at lines 171, 177, 173, 181 construct the messages which are then printed — we could have lifted the two calls to `calcStrPrefsCallMsg` to just one before the `if`

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with
Print Statements

String Prefixes with
Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

String Prefixes (6)

String Prefixes with Call, Return Messages

- ▶ The snag with inserting print statements into code is the program ceases to be a pure function — rather it is a program that takes a string and returns a computation which does some I/O and returns the factorial
- ▶ This conflates understanding the original pure function with understanding where the output comes from in code with side effects
- ▶ This can be written as a pure function which takes a number and returns a triple of the output string, and lists of call and return messages, and then decide how to present this

```
189 def funcStrPrefsWithMsgs(aStr) :  
190     return helpFuncStrPrefsWM(aStr, 1, "", [], [])
```

Agenda

Adobe Connect

Recursion

Recursion Examples —
Introduction
Factorial

String Prefixes Example

String Prefixes
Definition

String Prefixes with
Print Statements

String Prefixes with Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (7)

helpFuncStrPrefsWM()

```

192 def helpFuncStrPrefsWM(aStr, call, strAccum
193                          , callMsgList, retMsgList) :
194     callMsg = calcStrPrefsCallMsg(call, aStr)
195     if (aStr == "") :           # line (A)
196         retMsg = calcStrPrefsRetMsg(call, aStr, strAccum)
197         return (aStr + strAccum
198                , [callMsg] + callMsgList
199                , [retMsg] + retMsgList )
200     else :                      # line (B)
201         (pAccum, pCallMsgList, pRetMsgList) \
202         = helpFuncStrPrefsWM(aStr[:-1], call+1, strAccum
203                              , callMsgList, retMsgList)
204         retMsg = calcStrPrefsRetMsg(call, aStr, pAccum)
205         return (aStr + pAccum
206                , [callMsg] + pCallMsgList
207                , [retMsg] + pRetMsgList )

```

Agenda

Adobe Connect

Recursion

Recursion Examples —
Introduction
Factorial

String Prefixes Example

String Prefixes
Definition

String Prefixes with
Print Statements

String Prefixes with Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —
Summary
Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (7a)

helpFuncStrPrefsWM() Description

- ▶ `helpFuncStrPrefsWM()` description — the arguments are:
- ▶ `n` initialised to argument of `funcStrPrefsWithMsgs()`
- ▶ `call` the call number initialised to 1
- ▶ `strAccum` is the base case (initialised to "") — this argument is not needed but there may be generalisations
- ▶ `callMsgList`, the call messages list, initialised to an empty list
- ▶ `retMsgList`, the return messages list, initialised to an empty list
- ▶ The recursive call to `helpFuncStrPrefsWM()` at line 202 increments `call` — mimics a global variable
- ▶ The call to `calcStrPrefsCallMsg()` is now before the `if` at line 194 — constructs the call message
- ▶ The calls to `calcStrPrefsRetMsg()` at lines 196,204 construct the return message

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with Call, Return Messages

String Prefixes with

Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (8)

calcStrPrefsCallMsg(), calcStrPrefsRetMsg()

```
209 def calcStrPrefsCallMsg(call, aStr) :  
210     return ("Call_" + str(call)  
211            + "_of_function_with_aStr_" + pStr(aStr) )  
  
213 def calcStrPrefsRetMsg(call, aStr, acc) :  
214     return ("Return_call_" + str(call)  
215            + "_returning_" + pStr(aStr)  
216            + "_+" + pStr(acc) )
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Recursion Examples —
Introduction
Factorial

String Prefixes Example

String Prefixes
Definition

String Prefixes with
Print Statements

String Prefixes with
Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —
Summary
Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites & 48/202

String Prefixes (9)

Utility Code for Formatting Output

```
72 nl = "\n"
74 sp = " "
75 sp20 = sp.join(['']*21) # string of 20 spaces
77 def pStr(aStr) :
78     return ("\" + aStr + "\"")
```

```
228 def printStrPrefsWM(aStr) :
229     (strPrefsFinal, callMsgList, retMsgList) \
230     = funcStrPrefsWithMsgs(aStr)
231     print(nl.join(callMsgList)
232           + nl
233           + nl.join(reversed(retMsgList))
234           + nl + "strPrefs("
235           + pStr(aStr) + ")_=_ "
236           + pStr(strPrefsFinal) )
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with
Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites & 49/202

String Prefixes (9a)

Utility Code for Formatting Output — Description

- ▶ `n1`, `sp` are defined to make the code easier to read
- ▶ `join()` and `reversed()` are Python built-in functions
- ▶ The return messages list is reversed to get the messages in the same order as the print statement version — this might take some thinking
- ▶ `pStr()` is needed since `str("")` does not produce `''`

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with
Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (10)

String Prefixes with Call, Return Messages — Output

► Sample output from `printStrPrefswM("abc")`

```
Python3>>> printStrPrefswM("abc")
Call 1 of function with aStr = "abc"
Call 2 of function with aStr = "ab"
Call 3 of function with aStr = "a"
Call 4 of function with aStr = ""
Return call 4 returning "" + ""
Return call 3 returning "a" + ""
Return call 2 returning "ab" + "a"
Return call 1 returning "abc" + "aba"
strPrefswM("abc") = "abcaba"
Python3>>>
```

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with Call, Return Messages

String Prefixes with

Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (11)

String Prefixes with Evaluation Messages

- ▶ The evaluation messages can also be generated
- ▶ The code requires a bit more thought since the evaluation messages are also recursive

```
240 def funcStrPrefsEval(aStr) :  
241     return helpFuncStrPrefsEval(aStr, 1, "", [], [])
```

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with

Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (12)

helpFuncStrPrefsEval()

```
243 def helpFuncStrPrefsEval(aStr, call, strAccum
244                           , argList, evalMsgList) :
245     if (aStr == "") :                # line (A)
246         reasonStr = "#_by_line_(A)"
247         evalStr = pStr("")
248         evalMsg = calcStrPrefsEvalMsg(call, aStr
249                                       , argList, evalStr
250                                       , reasonStr)
251         return (aStr + strAccum
252               , [aStr] + argList, [evalMsg] + evalMsgList )
253     else :                            # line (B)
254         reasonStr = "#_by_line_(B)"
255         evalStr = "strPrefs(" + pStr(aStr[:-1]) + ")"
256         evalMsg = calcStrPrefsEvalMsg(call, aStr
257                                       , argList, evalStr
258                                       , reasonStr)
259         (pAccum, pArgList, pEvalMsgList) \
260         = helpFuncStrPrefsEval(aStr[:-1], call+1
261                               , strAccum, [aStr] + argList, evalMsgList)
262         return (aStr + pAccum
263               , pArgList, [evalMsg] + pEvalMsgList )
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

- Recursion Examples — Introduction
- Factorial

String Prefixes Example

- String Prefixes Definition

- String Prefixes with Print Statements

- String Prefixes with Call, Return Messages

- String Prefixes with Evaluation Messages

- Recursion Examples — Summary
- Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & 53/202

String Prefixes (12a)

helpFuncStrPrefsEval() Description

- ▶ `helpFuncStrPrefsEval()` description — the arguments are:
- ▶ `n` initialised to the argument of `funcStrPrefsEval()`
- ▶ `call` the call number initialised to 1
- ▶ `strAccum` is the base case (initialised to "") — this argument is not needed but there may be generalisations
- ▶ `argList`, the previous arguments list, initialised to an empty list — the previous arguments are needed to calculate the evaluation message
- ▶ `evalMsgList`, the evaluation messages list, initialised to an empty list
- ▶ The recursive call to `helpFuncStrPrefsEval()` at line 260 increments `call` — this mimics a global variable
- ▶ The calls to `calcStrPrefsEvalMsg()` at lines 248,256 construct the evaluation message (see description below)

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with

Call, Return Messages

String Prefixes with

Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (13)

calcStrPrefsEvalMsg()

```
265 def calcStrPrefsEvalMsg(call, aStr
266                             , argList, evalStr
267                             , reasonStr) :
268     if (argList == []) :
269         retMsg = ("Call_" + str(call)
270                 + "_->" + pStr(aStr)
271                 + "_+" + evalStr
272                 + nl + sp20 + reasonStr)
273         return retMsg
274     else :
275         nextStr = argList[0]
276         nextArgList = argList[1:]
277         nextEvalStr \
278             = "(" + pStr(aStr) + "_+" + evalStr + ")"
279         return calcStrPrefsEvalMsg(call, nextStr
280                                     , nextArgList, nextEvalStr
281                                     , reasonStr)
```

- Notice `calcStrPrefsEvalMsg()` is recursive unlike `calcStrPrefsCallMsg()`, `calcStrPrefsRetMsg()`

Agenda

Adobe Connect

Recursion

- Recursion Examples — Introduction
- Factorial

String Prefixes Example

- String Prefixes Definition
- String Prefixes with Print Statements
- String Prefixes with Call, Return Messages
- String Prefixes with Evaluation Messages

- Recursion Examples — Summary
- Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (13a)

calcStrPrefsEvalMsg() Description

- ▶ `calcStrPrefsEvalMsg()` description — the arguments are:
- ▶ `call` the current call number
- ▶ `n` the current argument
- ▶ `argList` the list of previous arguments
- ▶ `evalStr` the expression on the right hand side of the current operation (see print output below)
- ▶ `reasonStr` is the string identifying the branch of the `if` used
- ▶ If `argList` is empty, a string is returned
- ▶ Otherwise there is a recursive call to wrap the constructed string with the outer arguments (see print output below)
- ▶ It is easy to get the arguments in the wrong order — perhaps this is why this function is not given often.

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with

Call, Return Messages

String Prefixes with Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

String Prefixes (14)

Utility Code for Formatting Output

```
293 def printStrPrefsEval(aStr) :  
294     (strPrefsFinal, argList, evalMsgList) \  
295     = funcStrPrefsEval(aStr)  
296     print(nl.join(evalMsgList)  
297           + nl + "strPrefs("  
298           + pStr(aStr) + ")_=_"  
299           + pStr(strPrefsFinal) )
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with
Call, Return Messages

String Prefixes with
Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites & 57/202

String Prefixes (15)

String Prefixes with Evaluation Messages — Output

- Sample output from `printStrPrefsEval("abc")`

```
Python3>>> printStrPrefsEval("abc")
Call 1 -> "abc" + strPrefs("ab")
           # by line (B)
Call 2 -> "abc" + ("ab" + strPrefs("a"))
           # by line (B)
Call 3 -> "abc" + ("ab" + ("a" + strPrefs("")))
           # by line (B)
Call 4 -> "abc" + ("ab" + ("a" + (" " + " ")))
           # by line (A)
strPrefs("abc") = "abcaba"
Python3>>>
```

- Looks just like a hand evaluation

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

String Prefixes

Definition

String Prefixes with

Print Statements

String Prefixes with

Call, Return Messages

String Prefixes with Evaluation Messages

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Recursion Examples

Summary (1)

- ▶ The evaluation model of substituting actual arguments into the body of a function definition is the easiest for hand evaluation — it helps if there are no side effects or mutated variables
- ▶ The messages from print statements may be useful for seeing the pattern of calls or the state of a computation but requires some insight to understand the order of the output
- ▶ The examples used are similar to *M269 Python Activities* 3.2 `listSum()`, 3.3 `factorial()`, 3.4 `sumOddNumber()`

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Recursion Examples

Summary (2)

- ▶ Of course, a real functional programmer would take this a lot further — see [The Evolution of a Haskell Programmer](#)
- ▶ The point of this [piece of Haskell humour](#) is that *Recursion is the GOTO of Functional Programming* and you should really invent the appropriate higher order function to capture your pattern of recursion
- ▶ If you want to read further then have a look at [An Introduction to Recursion Schemes](#) though be aware that this becomes very quickly more advanced

Recursion Exercises

Evaluations

- ▶ The following exercises ask you to do evaluations of some expression involving a function with a recursive definition
- ▶ Do the evaluations in the same style as the above examples
- ▶ The Python code is in file [Recursion2020Exercises.py](#) in the same folder as the notes
- ▶ Note that some Python comments contain LaTeX code — this there to generate references
- ▶ The python refers to the [timeit](#) library — see [timeit](#) — [Measure execution time of small code snippets](#)

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Recursion Evaluations

Activity 1 fib01

- Evaluate `fib01(5)` given the following definition

```
17 def fib01(n) :  
18     if n == 0 :  
19         return 0          # fib01 (A)  
20     elif n == 1 :  
21         return 1          # fib01 (B)  
22     else :  
23         return (fib01(n - 2) + fib01(n - 1))  
24                     # fib01 (C)
```

- Refer to lines (A) or 19, (B) or 21, (C) or 23

► Go to Answer

Recursion Evaluations

Answer 1 fib01

```
# fib01(5) evaluation
Call 1 -> fib01(3) + fib01(4)           # by line (C)
Calls 2,3 -> (fib01(1) + fib01(2))
          + (fib01(2) + fib01(3))      # by line (C)x2
Calls 4,5,6,7 -> (1 + (fib01(0) + fib01(1)))
                + ((fib01(0) + fib01(1))
                  + (fib01(1) + fib01(2)))
          # by line (B),(C)x3
Calls 8-13 -> (1 + (0 + 1)))
              + ((0 + 1)
                + (1 + (fib01(0) + fib01(1))))
          # by line (A)x2,(B)x3,(C)
Calls 14-15 -> (1 + (0 + 1)))
               + ((0 + 1)
                 + (1 + (0 + 1)))
          # by line (A),(B)
fib01(5) = 5
```

- This takes 15 calls to `fib01()`, 5 calls to `fib01(1)`, 3 calls to `fib01(0)`

► Go to Activity

Agenda

Adobe Connect

Recursion

Recursion Examples —
Introduction

Factorial

String Prefixes Example

Recursion Examples —
Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Recursion Evaluations

Activity 2 fib02

- Evaluate `fib02(5)` given the following definition

```
28 def fib02(n) :  
29     if n == 0 :  
30         return 0          # fib02 (A)  
31     elif n == 1 :  
32         return 1          # fib02 (B)  
33     else :  
34         return (fib02List(n,0,1)[n])  
35                     # fib02 (C)  
  
37 def fib02List(n,first,second) :  
38     if n == 0 :  
39         return [first] # fib02List (D)  
40     else :  
41         return ([first  
42                 + fib02List(n - 1, second, first + second))  
43                 # fib02List (E)
```

► Go to Answer

Recursion Evaluations

Answer 2 fib02

```
# fib02(5) evaluation
Call 1 -> fib02List(5,0,1)[5]  # by line fib02 (C)
fib02List(5,0,1) -> ([0]      # by line fib02List (E)
                    + fib02List(4,1,1))
fib02List(4,1,1) -> ([1]      # by line fib02List (E)
                    + fib02List(3,1,2))
fib02List(3,1,2) -> ([1]      # by line fib02List (E)
                    + fib02List(2,2,3))
fib02List(2,2,3) -> ([2]      # by line fib02List (E)
                    + fib02List(1,3,5))
fib02List(1,3,5) -> ([3]      # by line fib02List (E)
                    + fib02List(0,5,8))
fib02List(0,5,8) -> [5]       # by line fib02List (D)
fib02List(5,0,1) = [0, 1, 1, 2, 3, 5]
fib02(5) = 5
```

- This takes 6 calls to **fib02List**

► Go to Activity

Fibonacci Sequence

Calculation (1)

- From [How to Compute Fibonacci Numbers?](#) and [Fibonacci Formulae](#)

$$\text{fib}(n + k) = \text{fib}(n-1) * \text{fib } k + \text{fib } n * \text{fib}(k+1)$$

- Proof by induction

$$\begin{aligned} \text{fib}(n + 2 + k) &= \text{fib}(n+k) + \text{fib}(n+k+1) && \text{-- by fib} \\ &= \text{fib}(n-1) * \text{fib } k + \text{fib } n * \text{fib}(k+1) \\ &\quad + \text{fib } n * \text{fib } k + \text{fib}(n+1) * \text{fib}(k+1) && \text{-- by hyp} \\ &= \text{fib}(n+1) * \text{fib } k + \text{fib}(n+2) * \text{fib}(k+1) && \text{-- by fib} \end{aligned}$$

Fibonacci Sequence

Calculation (2)

- We now derive a method to compute **fib** in $O(\log n)$

```
fib (2n+1) = (fib n)^2 + (fib (n+1))^2 -- (1)
fib (2n+2) = fib n * fib (n+1) + fib (n+1) * fib (n+2)
            = fib n * fib (n+1) + fib (n+1) * (fib n + fib (n+1))
            = 2 * fib n * fib (n+1) + (fib (n+1))^2 -- (2)
fib 2n = 2 * fib n * fib (n+1) - (fib n)^2 -- (3)
-- (3) = (2) - (1)
```

```
fib2v :: Integer -> (Integer,Integer)
fib2v 0 = (0,1)
fib2v n
  | n `mod` 2 == 0 = (c,d)
  | otherwise     = (d,c+d)
  where
    (a,b) = fib2v (n `div` 2)
    c      = 2 * a * b - a * a
    d      = a * a + b * b
```

Recursion Evaluations

Activity 3 fibv

- Evaluate `fibv(5)` given the following definition

```
def fibv(n) :  
    if n == 0 :  
        return 0          # fibv (A)  
    else :  
        return (fib2v(n - 1)[1])  
                        # fibv (B)  
  
def fib2v(n) :  
    if n == 0 :  
        return (0,1)      # fib2v (C)  
    else :  
        (a,b) = fib2v(n // 2)  
        c = 2 * a * b - a * a  
        d = a * a + b * b  
        if (n % 2 == 0) :  
            return (c,d) # fib2v (D)  
        else :  
            return (d, c + d)  
                    # fib2v (E)
```

► Go to Answer

Recursion Evaluations

Answer 3 fibv

```
# fibv(5) evaluation
fibv(5) -> fib2v(4)[1] # by fibv (B)
fib2v(4) -> (c,d)      # by fib2v (D)
  where (a,b) = fib2v(2)
    c = 2 * a * b - a * a
    d = a * a + b * b
fib2v(2) -> (c,d)      # by fib2v (D)
  where (a,b) = fib2v(1)
    c = 2 * a * b - a * a
    d = a * a + b * b
fib2v(1) -> (d, c + d) # by fib2v (E)
  where (a,b) = fib2v(0)
    c = 2 * a * b - a * a
    d = a * a + b * b
fib2v(0) -> (0,1)      # by fib2v (C)
-> fib2v(1) = (1,1)    # by arithmetic
-> fib2v(2) = (1,2)
-> fib2v(4) = (3,5)
-> fibv(5) = 5
```

► Go to Activity

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Recursion Examples —](#)

[Introduction](#)

[Factorial](#)

[String Prefixes Example](#)

[Recursion Examples —](#)

[Summary](#)

[Recursion Exercises](#)

[Sorting](#)

[Searching](#)

[String Search](#)

[Bags](#)

[Abstract Data Types](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic
Programming](#)

[Future Work](#)

[Web Sites &
References](#)

Recursion Evaluations

Example Timings — Python

```
Python3>>> timeit(lambda: fib01(40),number=1)
41.122885193966795
Python3>>> timeit(lambda: fib02(40),number=1)
3.125402145087719e-05
Python3>>> timeit('fib02(40)',globals=globals())
16.959519409982022
Python3>>> timeit('fib02(40)',globals=globals(),number=1)
2.4509034119546413e-05
Python3>>> timeit(lambda: fibv(40),number=1)
6.077397847548127e-05
Python3>>> timeit(lambda: fibv(200),number=1)
2.392695751041174e-05
Python3>>> timeit(lambda: fib02(200),number=1)
0.00042063603177666664
Python3>>> timeit(lambda: fibv(40),number=1)
7.555994670838118e-06
Python3>>> timeit(lambda: fibv(40),number=1)
7.733004167675972e-06
Python3>>> fib02(200)
280571172992510140037611932413038677189525
Python3>>> fibv(200)
280571172992510140037611932413038677189525
```

Agenda

[Adobe Connect](#)

Recursion

[Recursion Examples —](#)

[Introduction](#)

[Factorial](#)

[String Prefixes Example](#)

[Recursion Examples —](#)

[Summary](#)

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Web Sites & References

Recursion Evaluations

Example Timings — Haskell

```
GHCi> :set +s
GHCi> fib02 40
102334155
(0.00 secs, 89,456 bytes)
GHCi> fib02 200
280571172992510140037611932413038677189525
(0.00 secs, 186,992 bytes)
GHCi> fibv 200
280571172992510140037611932413038677189525
(0.00 secs, 104,528 bytes)
GHCi> fib01 40
^CInterrupted.
GHCi> fibv 300
222232244629420445529739893461909967206666939096499764990979600
(0.01 secs, 121,544 bytes)
```

M269 Units 3,4,5

Phil Molyneux

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Recursion Examples —](#)

[Introduction](#)

[Factorial](#)

[String Prefixes Example](#)

[Recursion Examples —](#)

[Summary](#)

[Recursion Exercises](#)

[Sorting](#)

[Searching](#)

[String Search](#)

[Bags](#)

[Abstract Data Types](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic
Programming](#)

[Future Work](#)

[Web Sites &
References](#)

Recursion Evaluations

Example Timings — Haskell (2)

```
GHCi> fib01 30
832040
(1.74 secs, 921,589,368 bytes)
GHCi> fib01 33
3524578
(7.31 secs, 3,903,684,264 bytes)
GHCi> fib01 34
5702887
(12.44 secs, 6,316,252,408 bytes)
GHCi> fib01 35
9227465
(19.20 secs, 10,219,870,536 bytes)
GHCi> fib01 36
14930352
(31.08 secs, 16,536,056,616 bytes)
GHCi> fib01 37
24157817
(49.99 secs, 26,755,861,504 bytes)
GHCi> fib01 38
39088169
(84.35 secs, 43,291,848,464 bytes)
```

Agenda

Adobe Connect

Recursion

Recursion Examples —

Introduction

Factorial

String Prefixes Example

Recursion Examples —

Summary

Recursion Exercises

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

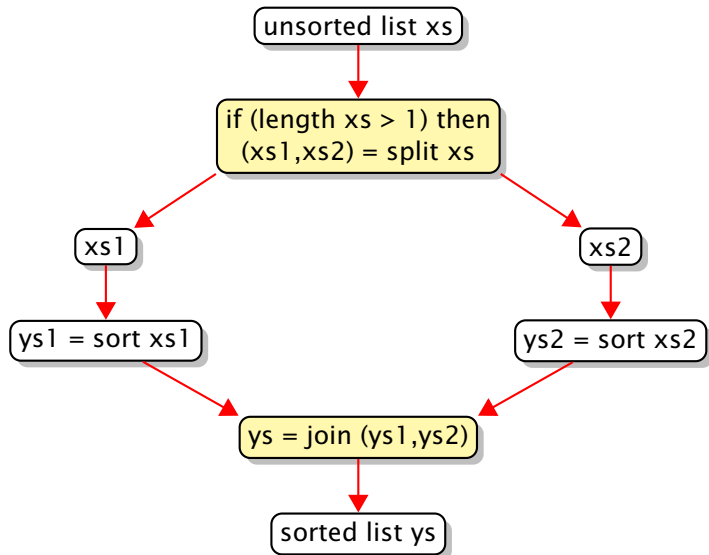
Dynamic Programming

Future Work

Web Sites & References

Taxonomy of Comparison Sorts

Abstract Sorting Algorithm

[Agenda](#)[Adobe Connect](#)[Recursion](#)[Sorting](#)[Taxonomy of Comparison Sorts](#)[Insertion Sort](#)[Searching](#)[String Search](#)[Bags](#)[Abstract Data Types](#)[Search Trees](#)[Graph Algorithms](#)[Greedy Algorithms](#)[Dynamic Programming](#)[Future Work](#)[Web Sites & References](#)

Sorting Algorithms

Other Classifications

- ▶ See [Wikipedia Sorting algorithm](#) for big list
- ▶ *Comparison Sorts*
 - ▶ Insertion sort, Selection sort, Merge sort, Quicksort, Bubble sort
 - ▶ Sorting via a data structure: Tree sort, Heap sort
- ▶ *Non-Comparison sorts* — distribution sorts — bucket sort, radix sort
- ▶ *Sorts used in Programming Language Libraries*
 - ▶ [Timsort](#) by Tim Peters — used in Python and Java — combination of merge and insertion sorts
 - ▶ [Haskell](#) — modified Mergesort by Ian Lynagh in [GHC](#) implementation

Insertion Sort

Abstract Algorithm

- ▶ *Insertion Split* **xs1** is the singleton list of the first item; **xs2** is the rest of the list
- ▶ *Insertion Join* insert the item in the singleton list into the sorted result of the rest of the list

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

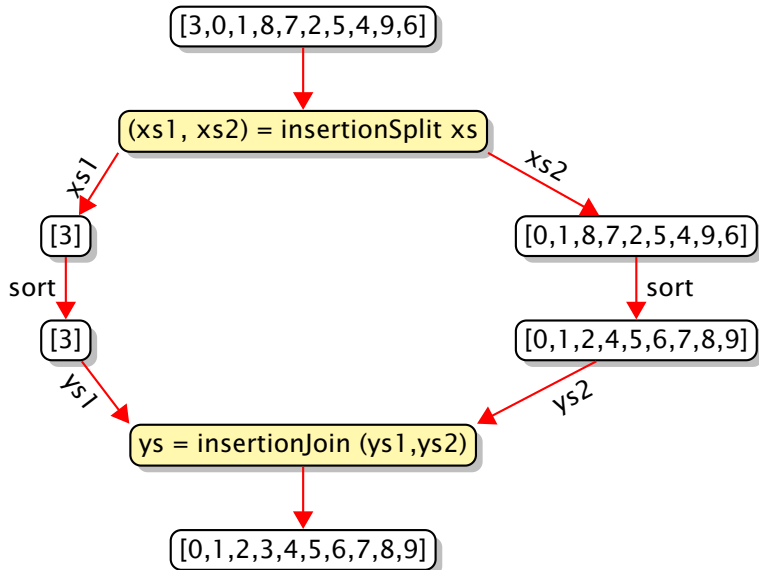
Greedy Algorithms

Dynamic Programming

Future Work

Insertion Sort

Abstract Sorting Algorithm Diagram



Insertion Sort

Python Implementation

```
4 def insSort(xs):
5     if len(xs) <= 1:
6         return xs
7     else:
8         return ins(xs[0], insSort(xs[1:]))
10 def ins(x, xs):
11     if xs == []:
12         return [x]
13     elif x <= xs[0]:
14         return [x] + xs
15     else:
16         return [xs[0]] + ins(x, xs[1:])
```

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Insertion Sort

Python Rewritten

► In the style of the abstract algorithm

```
20 def insSort01(xs):
21     if len(xs) <= 1:
22         return xs
23     else:
24         (xs1,xs2) = insertionSplit(xs)
25         ys1 = insSort01(xs1)
26         ys2 = insSort01(xs2)
27         ys = insertionJoin(ys1,ys2)
28         return ys

30 def insertionSplit(xs):
31     (xs1,xs2) = (xs[0:1],xs[1:])
32     return (xs1,xs2)

34 def insertionJoin(ys1,ys2):
35     if ys2 == []:
36         return ys1
37     elif ys1[0] <= ys2[0]:
38         return ys1 + ys2
39     else:
40         return ys2[0:1] + insertionJoin(ys1,ys2[1:])
```

Insertion Sort

Haskell Implementation (1)

```
1 module M269Prsntn2018JTTutorialUnits345 where
2   import Data.List
3   import Data.Maybe
```

1. A Haskell script starts with a module header which starts with the reserved identifier, `module` followed by the module name, `M269TutorialSorting`
2. The module name must start with an upper case letter and is the same as the file name (without its extension of `.lhs`)
3. Haskell uses *layout* (or the *off-side rule*) to determine scope of definitions, similar to Python
4. The body of the module follows the reserved identifier `where` and starts with two `import` declarations
5. These import the built-in libraries `Data.List` and `Data.Maybe`

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Insertion Sort

Haskell Implementation (2)

```
5 insSort [] = []
6 insSort [x] = [x]
7 insSort (x : xs) = ins x (insSort xs)

9 ins x [] = [x]
10 ins x (y:ys)
11   = if x <= y
12     then x:y:ys
13     else y : (ins x ys)
```

- ▶ For *structured English*, I have used a subset of *Haskell* (<http://haskell.org>) — in the code above:
- ▶ *insSort* and *ins* are function defined by several equations
- ▶ We use *indentation* to determine scope — see Landin (1966) and Python (see <http://docs.python.org/2/tutorial/introduction.html#first-steps-towards-programming> in the *Python Tutorial*)

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Insertion Sort

Haskell Implementation (3)

- ▶ Function application is denoted by juxtaposition and is more tightly binding than (almost) anything else
 - ▶ we write `f x` and not `f (x)`
 - ▶ `f x y` means `(f x) y`

This notational convention has huge advantages — discuss and also see

http://en.wikipedia.org/wiki/Curried_function and <http://slid.es/gsklee/functional-programming-in-5-minutes> (which does it in JavaScript, worth a look)

- ▶ Lists are denoted with brackets `[1,2,3]`, the empty list is `[]`
- ▶ `(:)` is the operator that prefixes an element to a list, `1:[2,3] == [1,2,3]`
- ▶ Parentheses over-ride precedence

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Activity 2

Trace an Evaluation — Haskell

- ▶ Evaluation of `insSort [3,0,1,8,7]`
- ▶ Answer goes here

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Activity 2

Trace an Evaluation — Haskell

► Evaluation of `insSort [3,0,1,8,7]`

Expression to Evaluate	Reason
<code>insSort [3,0,1,8,7]</code>	Initial
→ <code>ins 3 (insSort [0,1,8,7])</code>	line 7
→ <code>ins 3 (ins 0 (insSort [1,8,7]))</code>	line 7
→ <code>ins 3 (ins 0 (ins 1 (insSort [8,7])))</code>	line 7
→ <code>ins 3 (ins 0 (ins 1 (ins 8 (insSort [7])))</code>	line 7
→ <code>ins 3 (ins 0 (ins 1 (ins 8 [7])))</code>	line 6
→ <code>ins 3 (ins 0 (ins 1 (7:(ins 8 []))))</code>	line 13
→ <code>ins 3 (ins 0 (ins 1 (7:[8])))</code>	line 9
→ <code>ins 3 (ins 0 (ins 1 [7,8]))</code>	(:) operator
→ <code>ins 3 (ins 0 (1:7:[8]))</code>	line 12
→ <code>ins 3 (ins 0 [1,7,8])</code>	(:) operator
→ <code>ins 3 (0:1:[7,8])</code>	line 12
→ <code>ins 3 [0,1,7,8]</code>	(:) operator
→ <code>0:(ins 3 [1,7,8])</code>	line 13
→ <code>0:(1:(ins 3 [7,8]))</code>	line 13
→ <code>0:(1:(3:7:[8]))</code>	line 12
→ <code>[0,1,3,7,8]</code>	(:) operator

- Note that the evaluation consumes more space in the process of evaluation;
- also note that you need to be careful with the brackets when doing an evaluation like this by hand.

Activity 2 Trace an Evaluation

Insertion Sort — Python Recursive

- ▶ Evaluation of `insSort([3,0,1,8,7])`
- ▶ Answer goes here

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic Programming

Future Work

Activity 2 Trace an Evaluation

Insertion Sort — Python Recursive

► Evaluation of `insSort([3,0,1,8,7])`

Expression to Evaluate	Reason
<code>insSort([3,0,1,8,7])</code>	Initial
→ <code>ins(3, insSort([0,1,8,7]))</code>	line 7
→ <code>ins(3, ins(0, insSort([1,8,7])))</code>	line 7
→ <code>ins(3, ins(0, ins(1, insSort([8,7])))</code>	line 7
→ <code>ins(3, ins(0, ins(1, ins(8, insSort([7])))))</code>	line 7
→ <code>ins(3, ins(0, ins(1, ins(8, [7]))))</code>	line 5
→ <code>ins(3, ins(0, ins(1, ([7] + ins(8, []))))</code>	line 15
→ <code>ins(3, ins(0, ins(1, ([7] + [8])))</code>	line 11
→ <code>ins(3, ins(0, ins(1, [7,8])))</code>	(+) operator
→ <code>ins(3, ins(0, ([1] + [7,8])))</code>	line 13
→ <code>ins(3, ins(0, [1,7,8]))</code>	(+) operator
→ <code>ins(3, ([0] + [1,7,8]))</code>	line 13
→ <code>ins(3, [0,1,7,8])</code>	(+) operator
→ <code>[0] + (ins 3 [1,7,8])</code>	line 15
→ <code>[0] + ([1] + (ins 3 [7,8]))</code>	line 15
→ <code>[0] + ([1] + ([3] + ([7,8])))</code>	line 13
→ <code>[0,1,3,7,8]</code>	(+) operator

- Note that the evaluation consumes more space in the process of evaluation;
- also note that you need to be careful with the brackets when doing an evaluation like this by hand.

Insertion Sort

Non-recursive Implementation

- ▶ The non-recursive version of *Insertion* sort takes each element in turn and inserts it in the ordered list of elements before it.

```
for index = 1 to (len(xs)-1) do
  insert xs[index] in order in xs[0..index-1]
```

- ▶ Here is a Python implementation of the above (based on Miller and Ranum (2011, page 215)).

```
44 def insertionSort(xs):
45     for index in range(1, len(xs)):
46         currentValue = xs[index]
47         position = index
48         while (position > 0) and xs[position - 1] > currentValue:
49             xs[position] = xs[position - 1]
50             position = position - 1
51
52     xs[position] = currentValue
```

Activity 3

Trace an Evaluation — Python Non-recursive

- ▶ Evaluation of `insertionSort([3,0,1,8,7])`
- ▶ Showing just the outer `for index` loop

▶

3	0	1	8	7
---	---	---	---	---

 start array

- ▶ Answer goes here

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Activity 3

Trace an Evaluation — Python Non-recursive

► Evaluation of `insertionSort([3,0,1,8,7])`

► Showing just the outer `for` `index` loop

►

3	0	1	8	7
---	---	---	---	---

 start array

3	0	1	8	7
---	---	---	---	---

 index = 1

0	3	1	8	7
---	---	---	---	---

 index = 2

0	1	3	8	7
---	---	---	---	---

 index = 3

0	1	3	8	7
---	---	---	---	---

 index = 4

0	1	3	7	8
---	---	---	---	---

 end

Agenda

Adobe Connect

Recursion

Sorting

Taxonomy of
Comparison Sorts

Insertion Sort

Insertion Sort —
Abstract Algorithm

Insertion Sort — Python

Insertion Sort —
Haskell

Activity 2 — Insertion
Sort: Trace an
Evaluation

Insertion Sort —
Non-recursive

Activity 3 — Insertion
Sort Non-recursive
Trace

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Binary Search

Example Algorithm Design

- ▶ Given an ordered list (*xs*) and a value (*val*), return
 - ▶ Position of *val* in *xs* or
 - ▶ Some indication if *val* is not present
- ▶ *Simple strategy*: check each value in the list in turn
- ▶ *Better strategy*: use the ordered property of the list to reduce the range of the list to be searched each turn
 - ▶ Set a range of the list
 - ▶ If *val* equals the mid point of the list, return the mid point
 - ▶ Otherwise half the range to search
 - ▶ If the range becomes negative, report not present (return some distinguished value)
- ▶ *Binary Search* in various forms

Binary Search

Binary Search Iterative

```
5 def binarySearchIter(xs, val):
6     lo = 0
7     hi = len(xs) - 1
8
9     while lo <= hi:
10         mid = (lo + hi) // 2
11         guess = xs[mid]
12
13         if val == guess:
14             return mid
15         elif val < guess:
16             hi = mid - 1
17         else:
18             lo = mid + 1
19
20     return None
```

M269 Units 3,4,5

Phil Molyneux

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Sorting](#)

[Searching](#)

[Binary Search](#)

[Binary Search Exercise](#)

[Binary Search Error](#)

[Binary Search —
Position](#)

[String Search](#)

[Bags](#)

[Abstract Data Types](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic
Programming](#)

[Future Work](#)

[Web Sites &
References](#)

Binary Search

Binary Search Recursive

```
24 def binarySearchRec(xs, val, lo=0, hi=-1):
25     if (hi == -1):
26         hi = len(xs) - 1
27
28     mid = (lo + hi) // 2
29
30     if hi < lo:
31         return None
32     else:
33         guess = xs[mid]
34         if val == guess:
35             return mid
36         elif val < guess:
37             return binarySearchRec(xs, val, lo, mid-1)
38         else:
39             return binarySearchRec(xs, val, mid+1, hi)
```

M269 Units 3,4,5

Phil Molyneux

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Sorting](#)

[Searching](#)

[Binary Search](#)

[Binary Search Exercise](#)

[Binary Search Error](#)

[Binary Search —
Position](#)

[String Search](#)

[Bags](#)

[Abstract Data Types](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic
Programming](#)

[Future Work](#)

[Web Sites &
References](#)

Example Algorithm Design

Binary Search — Exercise

Given the *Python* definition of `binarySearchRec` from above, trace an evaluation of `binarySearchRec(xs, 25)` where `xs` is

`xs = [2, 5, 7, 15, 17, 19, 21, 24, 27, 31, 37, 48, 57, 87, 95]`

Example Algorithm Design

Binary Search — Solution

xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]

binarySearchRec(xs, 25)

XS = Highlight the *mid value* and *search range*

binarySearchRec(xs,25,??,??)

XS = Highlight the *mid value* and *search range*

binarySearchRec(xs,25,??,??)

XS = Highlight the *mid value* and *search range*

binarySearchRec(xs,25,??,??)

XS = Highlight the *mid value* and *search range*

binarySearchRec(xs,25,??,??)

Return value: ??

Example Algorithm Design

Binary Search — Solution

xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]

binarySearchRec(xs, 25)

xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]

binarySearchRec(xs,25,??,??)

XS = Highlight the mid value and search range

binarySearchRec(xs,25,??,??)

XS = Highlight the mid value and search range

binarySearchRec(xs,25,??,??)

XS = Highlight the mid value and search range

binarySearchRec(xs,25,??,??)

Return value: ??

Example Algorithm Design

Binary Search — Solution

xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]

binarySearchRec(xs, 25)

xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]

binarySearchRec(xs,25,8,14) *by line 38*

XS = *Highlight the mid value and search range*

binarySearchRec(xs,25,??,??)

XS = *Highlight the mid value and search range*

binarySearchRec(xs,25,??,??)

XS = *Highlight the mid value and search range*

binarySearchRec(xs,25,??,??)

Return value: ??

Example Algorithm Design

Binary Search — Solution

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs, 25)
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,14) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,??,??)
XS = Highlight the mid value and search range
binarySearchRec(xs,25,??,??)
XS = Highlight the mid value and search range
binarySearchRec(xs,25,??,??)
Return value: ??
```

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Example Algorithm Design

Binary Search — Solution

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs, 25)
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,14) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 38
XS = Highlight the mid value and search range
binarySearchRec(xs,25,??,??)
XS = Highlight the mid value and search range
binarySearchRec(xs,25,??,??)
Return value: ??
```

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Example Algorithm Design

Binary Search — Solution

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs, 25)
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,14) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,??,??)
XS = Highlight the mid value and search range
binarySearchRec(xs,25,??,??)
Return value: ??
```

Example Algorithm Design

Binary Search — Solution

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs, 25)
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,14) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 36
XS = Highlight the mid value and search range
binarySearchRec(xs,25,??,??)
Return value: ??
```

Example Algorithm Design

Binary Search — Solution

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs, 25)
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,14) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 36
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,??,??)
Return value: ??
```

Example Algorithm Design

Binary Search — Solution

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs, 25)
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,14) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 36
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,7) by line 36
Return value: ??
```

Example Algorithm Design

Binary Search — Solution

```
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs, 25)
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,14) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 38
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 36
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,7) by line 36
Return value: None by line 30
```

Binary Search

Comparison of Iterative and Recursive Functions

- ▶ Both forms compare the given value (`val`) to the mid-point value of the range of the list (`xs[mid]`)
- ▶ If not found, the range is adjusted via assignment in a `while` loop (iterative) or function call (recursive)
- ▶ The recursive version has *default parameter* values to initialise the function call (evil, should be a helper function)
- ▶ There are two base cases:
 - ▶ The value is found (`val == guess`)
 - ▶ The range becomes negative (`hi < lo`)
- ▶ The return value is either `mid` or `None`
- ▶ What is the *type* of the binary search function ?
- ▶ How do we *reason* about the code ?

Binary Search

Example Error

- ▶ Although the basic idea of binary search is comparatively straightforward, the details can be surprisingly tricky
(Donald Knuth, 1998, The Art of Computer Programming Vol. 3: Sorting and Searching, section 6.2.1)
- ▶ The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.
(Donald Knuth, 1974, Computer Programming as an Art: 1974 Turing Award Lecture, CACM)
- ▶ See other Donald Knuth quotes at [Wikiquote: Donald Knuth](#)

Binary Search

Version with Error

```
46 def binarySearchError01(xs, target):
47     lo = 0
48     hi = len(xs)
50     while True :
51         centre = (hi + lo)//2
52         if centre == lo :
53             # Only two items left to test so it is either
54             # centre or centre + 1 or it is not in. This is
55             # an exit point of the infinite loop
56             return (centre if xs[centre] == target
57                     else centre+1 if xs[centre+1] == target
58                     else -1)
59         if xs[centre] < target :
60             lo = centre
61         elif target < xs[centre] :
62             hi = centre
63         else :
64             return centre
```

Binary Search

Error Exercise

- ▶ Suppose `xs = [10,20]`, `target = 30`
- ▶ Trace the execution of the above code and identify a run time error
- ▶ This is exercise 4.6 in Backhouse (2003, pages 50–51, 269–270) which refers to the text Winder and Roberts (1998, page 592), which was repeated in Winder and Roberts (2000, page 561)

Binary Search

Error Exercise

	lo	hi	centre	xs[centre]
Before loop	0	2	-	-
Loop 1	0	2	1	20
centre == lo	False			
xs[centre] < target	True			
Loop 2	1	2	1	20
centre == lo	True			
Crash				

- ▶ `xs = [10,20], target = 30`
- ▶ And we get an out of range error
- ▶ How should it be corrected ?
- ▶ Throw it away and start again

Binary Search

Position

- ▶ M269 2017J TMA02 Q1 provides an iterative function to return the position where a target is or should be inserted.
- ▶ Insert 25 into [10,20,30] should return [10,20,25,30]
- ▶ Insert 20 into [10,20,30] should return [10,20,20,30]
- ▶ Insert 5 into [10,20,30] should return [5,10,20,30]
- ▶ Insert 35 into [10,20,30] should return [10,20,30,35]

Binary Search

Position

- ▶ TMA02 Q1(b) specifies the function, **binSchPosnIterTMA** to return the position
- ▶ **Inputs** xs sequence of n integers $x_0, x_1, \dots, x_{n-1}$
- ▶ $(first, last)$ index of first and last items in sorted subsequence
- ▶ $target$ integer to find insertion position
- ▶ **Pre** $0 \leq first < n$ and $last < n$
- ▶ If $first \leq last$ then $x_{first} \leq x_{first+1} \leq \dots \leq x_{last}$
- ▶ **Post** If $first > last$ then return value, pos , is $first$
- ▶ If $first \leq last$ and $target > x_{last}$ then $pos = last + 1$
- ▶ If $first \leq last$ and $target \leq x_{last}$ then pos is the smallest value such that $first \leq pos \leq last$ and $target \leq x_{pos}$

Binary Search — Position

TMA02 Q1(b) Code

```
68 def binSchPosnIterTMA(xs,first,last,target) :  
69     while first <= last :  
70         middle = (first + last) // 2  
71         if xs[middle] == target :  
72             return middle  
73         if target < xs[middle] :  
74             last = middle - 1  
75         else :  
76             first = middle + 1  
77     return first
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

TMA02 Q1(b) Code — Exercises (1)

► Evaluate

- `binSchPosnIterTMA([10,20,30],0,2,25)`
- `binSchPosnIterTMA([10,20,30],0,2,20)`
- `binSchPosnIterTMA([10,20,30],0,2,5)`
- `binSchPosnIterTMA([10,20,30],0,2,35)`

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

TMA02 Q1(b) Code — Exercises (1) — Answers

```
Python3>>> binSchPosnIterTMA([10,20,30],0,2,25)
2
Python3>>> binSchPosnIterTMA([10,20,30],0,2,20)
1
Python3>>> binSchPosnIterTMA([10,20,30],0,2,5)
0
Python3>>> binSchPosnIterTMA([10,20,30],0,2,35)
3
Python3>>>
```

► Are these tests enough ?

Binary Search — Position

TMA02 Q1(b) Code — Exercises (2)

- ▶ What if we have sequences of items with the same key as the target ?
- ▶ **Evaluate**
- ▶ `binSchPosnIterTMA([10,20,30],0,2,20)`
- ▶ `binSchPosnIterTMA([10,20,20,30],0,3,20)`
- ▶ `binSchPosnIterTMA([10,20,20,20,30],0,4,20)`
- ▶ `binSchPosnIterTMA([10,20,20,20,20,30],0,5,20)`

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

TMA02 Q1(b) Code — Exercises (2) — Answers

```
Python3>>> xs2
[10, 20, 20, 30]
Python3>>> binSchPosnIterTMA(xs2,0,3,20)
1
Python3>>> xs3
[10, 20, 20, 20, 30]
Python3>>> binSchPosnIterTMA(xs3,0,4,20)
2
Python3>>> xs4
[10, 20, 20, 20, 20, 30]
Python3>>> binSchPosnIterTMA(xs4,0,5,20)
2
Python3>>> xs5
[10, 20, 20, 20, 20, 20, 30]
Python3>>> binSchPosnIterTMA(xs5,0,6,20)
3
Python3>>>
```

- ▶ Does this meet the specification ?
- ▶ Why not ?

Binary Search — Position

Out by One Errors

- ▶ *Out by One* errors are common in implementation of Binary Search (or similar errors)
- ▶ The specification is best thought about as partitioning the original list so the insertion point is below or to the left of any identical entries
- ▶ So we write the postcondition to say that:
- ▶ $x_0, x_1, \dots, x_{pos-1} < \text{target}$
- ▶ $x_{pos}, \dots, x_{n-1} \geq \text{target}$
- ▶ where n is the length of the subsequence and pos is the insertion point returned by the function
- ▶ Getting any of the above numbers *out by one* or something similar, leads to bugs

Binary Search — Position

Loop Invariant

- ▶ To find *pos* we are going to maintain two parts of the subsequence
- ▶ $x_0, x_1, \dots, x_{\text{first}-1} < \text{target}$
- ▶ $x_{\text{last}+1}, \dots, x_{n-1} \geq \text{target}$
- ▶ We start with *first* as 0 and *last* as ($n - 1$)
- ▶ So at the start the two partitions are empty (and hence the conditions above are True)
- ▶ The **while** loop maintains the loop invariant above, while moving *first* and *last*
- ▶ At the end of the loop, $\text{last} = \text{first} - 1$
- ▶ Hence (with some arithmetic) we can check the post condition
- ▶ Below is **binSchPosnIterBelow** which implements this

Binary Search — Position

Insertion Point Below Sequence of Identical Items

```
81 def binSchPosnIterBelow(xs, first, last, target) :  
83     while first <= last :  
84         # Invariant:  
85         # Items in xs[:first] < target  
86         # Items in xs[last+1:] >= target  
87         middle = (first + last) // 2  
88         if xs[middle] < target :  
89             first = middle + 1  
90         else :  
91             last = middle - 1  
93     return first
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

Testing binSchPosnIterBelow()

```
Python3>>> xs2
[10, 20, 20, 30]
Python3>>> binSchPosnIterBelow(xs2,0,3,20)
1
Python3>>> xs3
[10, 20, 20, 20, 30]
Python3>>> binSchPosnIterBelow(xs3,0,4,20)
1
Python3>>> xs4
[10, 20, 20, 20, 20, 30]
Python3>>> binSchPosnIterBelow(xs4,0,5,20)
1
Python3>>> xs5
[10, 20, 20, 20, 20, 20, 30]
Python3>>> binSchPosnIterBelow(xs5,0,6,20)
1
Python3>>>
```

► Does this meet the specification ?

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

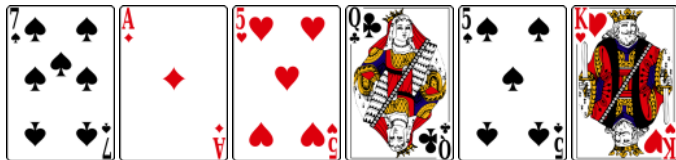
Future Work

Web Sites &
References

Binary Search — Position

Insertion Sort is a *stable* sort

- ▶ Insertion sort is a *stable* sort
- ▶ Items with identical keys are sorted in the same order as they appear in the input



- ▶ If the above cards are sorted by value as the key (ignoring suit) then the 5 of Spades should stay above (or to the right) of the 5 of Hearts
- ▶ `binSchPosnIterBelow()` would not do that
- ▶ How should we modify the algorithm to produce `binSchPosnIterAbove()` ?

Binary Search — Position

Insertion Point Above Sequence of Identical Items

```
97 def binSchPosnIterAbove(xs, first, last, target) :  
99     while first <= last :  
100         # Invariant:  
101         # Items in xs[:first] <= target  
102         # Items in xs[last+1:] > target  
103         middle = (first + last) // 2  
104         if xs[middle] <= target :  
105             first = middle + 1  
106         else :  
107             last = middle - 1  
109     return first
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

Testing binSchPosnIterAbove()

```
Python3>>> xs2
[10, 20, 20, 30]
Python3>>> binSchPosnIterAbove(xs2,0,3,20)
3
Python3>>> xs3
[10, 20, 20, 20, 30]
Python3>>> binSchPosnIterAbove(xs3,0,4,20)
4
Python3>>> xs4
[10, 20, 20, 20, 20, 30]
Python3>>> binSchPosnIterAbove(xs4,0,5,20)
5
Python3>>> xs5
[10, 20, 20, 20, 20, 20, 30]
Python3>>> binSchPosnIterAbove(xs5,0,6,20)
6
Python3>>>
```

- Does this meet the *new* specification ?

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

Insertion Point Below Sequence of Identical Items — Recursive

- ▶ As before the function must be called with a sorted subsequence of *xs* with *first* and *last* being the indices of the sorted part start and end
- ▶ Depending on the relation of `xs[middle]` to `target` we recurse on either the beginning or ending half of the subsequence
- ▶ The base case is where `first` and `last` cross over
- ▶ Notice how the recursive version follows the loop invariant

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

Insertion Point Below Sequence of Identical Items — Recursive

```
113 def binSchPosnRecBelow(xs, first, last, target) :  
115     middle = (first + last) // 2  
  
117     if first > last :  
118         return first  
119     elif xs[middle] < target :  
120         return binSchPosnRecBelow(xs, middle+1, last, target)  
121     else :  
122         return binSchPosnRecBelow(xs, first, middle-1, target)
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

Testing binSchPosnRecBelow()

```
Python3>>> xs2
[10, 20, 20, 30]
Python3>>> binSchPosnRecBelow(xs2,0,3,20)
1
Python3>>> xs3
[10, 20, 20, 20, 30]
Python3>>> binSchPosnRecBelow(xs3,0,4,20)
1
Python3>>> xs4
[10, 20, 20, 20, 20, 30]
Python3>>> binSchPosnRecBelow(xs4,0,5,20)
1
Python3>>> xs5
[10, 20, 20, 20, 20, 20, 30]
Python3>>> binSchPosnRecBelow(xs5,0,6,20)
1
Python3>>>
```

► Does this meet the specification ?

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Binary Search — Position

Insertion Point Above Sequence of Identical Items — Recursive

- ▶ We have the same small change to the function to produce the **Above** version
- ▶ Note that small changes to the function can make a big difference

```
126 def binSchPosnRecAbove(xs, first, last, target) :  
128     middle = (first + last) // 2  
130     if first > last :  
131         return first  
132     elif xs[middle] <= target :  
133         return binSchPosnRecAbove(xs, middle+1, last, target)  
134     else :  
135         return binSchPosnRecAbove(xs, first, middle-1, target)
```

- ▶ Note that the TMA02 Q1 version is slightly different to both the **Below** and **Above** versions

Binary Search — Position

Testing binSchPosnRecAbove()

```
Python3>>> xs2
[10, 20, 20, 30]
Python3>>> binSchPosnRecAbove(xs2,0,3,20)
3
Python3>>> xs3
[10, 20, 20, 20, 30]
Python3>>> binSchPosnRecAbove(xs3,0,4,20)
4
Python3>>> xs4
[10, 20, 20, 20, 20, 30]
Python3>>> binSchPosnRecAbove(xs4,0,5,20)
5
Python3>>> xs5
[10, 20, 20, 20, 20, 20, 30]
Python3>>> binSchPosnRecAbove(xs5,0,6,20)
6
Python3>>>
```

- Does this meet the *new* specification ?

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

Binary Search

Binary Search Exercise

Binary Search Error

Binary Search —
Position

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

String Search

- ▶ *Sunday Quick Search* algorithm
- ▶ *Knuth-Morris-Pratt (KMP)* algorithm
- ▶ **Exact String Matching Algorithms** — animations in Java

Sunday Quick Search Algorithm

Example

- ▶ Sunday Quick Search example
- ▶ Consider the alphabet C, G, A, T and a target string CGTACTCGTAGT.
- ▶ Calculate the shift table for this search, explaining in detail how you used the part of the algorithm that builds the table to derive your results.
- ▶ Given the target string CGTACTCGTAGT, the search string CGTACTCGGCGTAAAGTGCGTCTT and the shift table calculated above
- ▶ describe, with diagrams if necessary, how the first attempt to match the target string against the search string fails
- ▶ and how the target string slides along the search string for the second matching attempt.

Sunday Quick Search Algorithm

Sunday Quick Search Shift Table

- ▶ The shift table for the Sunday Quick Search algorithm:
 - ▶ If the character does not appear in the target string T , the shift distance is one more than the length of T
 - ▶ If the character does appear in T the shift distance is the first position at which it appears, counting from right to left and starting at 1
- ▶ Shift table:

A	C	G	T
3	6	2	1
- ▶ See the example at [Quick Search algorithm](#)

[Agenda](#)
[Adobe Connect](#)
[Recursion](#)
[Sorting](#)
[Searching](#)
[String Search](#)
[Sunday Quick Search Algorithm](#)
[Knuth-Morris-Pratt \(KMP\) Algorithm](#)
[Bags](#)
[Abstract Data Types](#)
[Search Trees](#)
[Graph Algorithms](#)
[Greedy Algorithms](#)
[Dynamic Programming](#)
[Future Work](#)
[Web Sites & References](#)

Sunday Quick Search Algorithm

Calculating the Shift

- Given the following alignment of *Search* and *Target* strings

Search string

C	G	T	A	C	T	C	G	G	C	G	T	A	A	G	T	G	C	G	T	C	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

C	G	T	A	C	T	C	G	T	A	G	T
---	---	---	---	---	---	---	---	---	---	---	---

Target string

- The next character in the *Search* string after the *Target* string is **A** so the shift will be 3 places

Search string

C	G	T	A	C	T	C	G	G	C	G	T	A	A	G	T	G	C	G	T	C	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

C	G	T	A	C	T	C	G	T	A	G	T
---	---	---	---	---	---	---	---	---	---	---	---

Target string

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Sunday Quick Search
Algorithm

Knuth-Morris-Pratt (KMP)
Algorithm

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Knuth-Morris-Pratt (KMP) Algorithm

Example

- ▶ Knuth-Morris-Pratt (KMP) algorithm example
- ▶ Consider the alphabet C, G, A, T and a target string CGTACTCGTAGT.
- ▶ Calculate the prefix table for this target string,
- ▶ explaining in detail how you derived the sixth, seventh and eighth entries in your table.

Knuth-Morris-Pratt (KMP) Algorithm

KMP Prefix Table

- ▶ The KMP *prefix table* takes the *target string*, t , and a *position*, p , in t and returns an integer k
- ▶ k is the length of the maximum proper prefix of t up to position p that is a suffix of t up position p
- ▶ We define a prefix function to take a *target string*, t , and a number $k \geq 0$, which is the number of items off the front of t
- ▶ Formally, if our position index is 0 based we have:
- ▶ $\text{prefixTable}(t, 0) = 0$
- ▶ $\text{prefixTable}(t, p)$
 $= \max\{k : k \leq p \wedge [\text{prefix}(t, k) \sqsubset \text{prefix}(t, p + 1)]\}$
- ▶ Here $\sqsubset$ means *proper suffix* — that is a suffix that does not include the whole string (Cormen, 2009, page 986)

Knuth-Morris-Pratt (KMP) Algorithm

KMP Shift Function

- ▶ M269 does not give the KMP shift function (or table) — we give it here for interest
- ▶ The shift function takes the *target string*, t , and the number of characters in the target string that have been matched, q and returns the shift.
- ▶ $\text{shift}(t, 0) = 1$
- ▶ $\text{shift}(t, q) = q - \text{prefixTable}(t, q - 1)$
- ▶ This assumes 0 based list indexing
- ▶ The notation here comes from Cormen et al (2009, section 32.4, page 1004) — this uses 1 based list indexing and also provides code.
- ▶ Cormen uses the terminology *Text* for *Search string* and *Pattern* for *Target string*

Knuth-Morris-Pratt (KMP) Algorithm

KMP Prefix and Shift Table

C	G	T	A	C	T	C	G	T	A	G	T
---	---	---	---	---	---	---	---	---	---	---	---

Target string

0	1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	---	----	----

Position (Index), p

0	1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	---	----	----	----

Match, q

0	0	0	0	1	0	1	2	3	4	0	0
---	---	---	---	---	---	---	---	---	---	---	---

$\text{prefixTable}(t, p)$

1	1	2	3	4	4	6	6	6	6	6	11	12
---	---	---	---	---	---	---	---	---	---	---	----	----

$\text{shift}(t, q)$

Bags

Definitions

- ▶ **Bag** unordered collection that may contain duplicate items
- ▶ Also known as **Multiset**
- ▶ **Multiplicity** of an element is the number of instances of an element
- ▶ A Bag or Multiset may be defined as a two-tuple (A, m) where A is the underlying set from which elements are drawn, and a function $m : A \rightarrow \mathbb{N}$
- ▶ Note that some definitions exclude 0 from the range of m so it would be denoted $m : A \rightarrow \mathbb{N}_{\geq 1}$

Bags

Example

- ▶ 120 has prime factorization $120 = 2^3 \times 3^1 \times 5^1$
- ▶ Gives bag $\{2, 2, 2, 3, 5\}$
- ▶ The $\{\}$ is an abuse of the usual set notation
- ▶ Representations:
- ▶ Sorted list $[2, 2, 2, 3, 5]$
- ▶ Unsorted list $[2, 3, 2, 5, 2]$
- ▶ Sorted list of pairs $[(2, 3), (3, 1), (5, 1)]$

Bags

Operations

- ▶ **Create** an empty bag
- ▶ **Queries**
 - ▶ `isEmptyBag`
 - ▶ `sizeBag` total number of elements
 - ▶ `elemOccurs` number of an element
 - ▶ `elemMember` is there at least one copy of an element
- ▶ **Construction**
 - ▶ `insertElem` insert one copy of an element
 - ▶ `insertMany` insert a number of copies
 - ▶ `deleteElem` delete a single copy of an element
 - ▶ `deleteMany` delete a number of copies
 - ▶ `deleteAll` deleteAll all copies

Bags

M269 2018J TMA02 Bag Operations

- ▶ `Bag()` creates a new empty bag
- ▶ `add(self, elem)` adds one copy of `elem` to the bag `self`
- ▶ `count(self, elem)` number of `elem` in the bag `self`
- ▶ `size(self)` total number of copies in `self`
- ▶ `clear(self, elem)` removes all copies of `elem`
- ▶ `ordered(self)` returns contents of bag `self` as a list of pairs `(count, elem)` in decreasing order of `count`

Bags

Implementations

- ▶ **Python Counter** is a subclass of **dict** which is Python's version of bags or multisets
- ▶ **Data.MultiSet** is Haskell's implementation of multisets or bags — it is based on **Data.Map**
- ▶ **Multiset** describes the ADT multiset or bag in several programming languages

Bags

Python dict

- ▶ A mapping object or dictionary maps hashable values to arbitrary objects
- ▶ A dictionary is a mutable object
- ▶ **Creation**

```
aD = dict()      # dictionary constructor
aD = {}          # literal empty dictionary
aD = {'to': 2, 'be': 2, 'or': 1, 'not': 1}
               # literal key:value pairs
```

Python dict

Creation

► Creation methods

```
Python3>>> aD = {'to': 2, 'be': 2
...              , 'or': 1, 'not': 1}
Python3>>> bD = dict(to=2, be=2, orA=1, notA=1)
Python3>>> cD = dict(zip(['to', 'be', 'or', 'not']
...                      , [2, 2, 1, 1]))
Python3>>> dD = dict([('to', 2), ('be', 2)
...                  , ('or', 1), ('not', 1)])
Python3>>> eD = dict({'to': 2, 'be': 2
...                  , 'or': 1, 'not': 1})
Python3>>> aD == cD == dD == eD
True
```

- Keywords in keyword arguments must not clash with built-in keywords
- **Implicit line joining** means we can split expressions in parentheses, square brackets or curly braces over more than one physical line without using backslashes.

Python dict

Queries

► Queries in the *M269 Companion*

```
Python3>>> aD = {'to': 2, 'be': 2
...              , 'or': 1, 'not': 1}
Python3>>> len(aD)
4
Python3>>> key = 'be'
Python3>>> key in aD
True
Python3>>> aD[key]
2
Python3>>> aD.keys()
dict_keys(['to', 'or', 'not', 'be'])
Python3>>> list(aD.keys())
['to', 'or', 'not', 'be']
Python3>>> aD.items()
dict_items([('to', 2), ('or', 1)
            , ('not', 1), ('be', 2)])
Python3>>> list(aD.items())
[('to', 2), ('or', 1), ('not', 1), ('be', 2)]
Python3>>> ('to', 2) in aD.items()
True
```

Dictionaries

Activity 4 Total Exercise

- Write a function, `totalValue`, that takes a dictionary, `xD`, and returns the total of all the values of the key:value pairs (Assumes value is a numeric type)

► Go to Answer

M269 Units 3,4,5

Phil Molyneux

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Sorting](#)

[Searching](#)

[String Search](#)

[Bags](#)

[Bags: Definitions](#)

[Bags: Implementations](#)

[Abstract Data Types](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic Programming](#)

[Future Work](#)

[Web Sites & References](#)

Dictionaries

Answer 4 Total Exercise

► Answer 4 Total Exercise

```
def totalValue(xD) :  
    """  
    totalValue takes a dictionary, xD,  
    and returns the total of all the values  
    of the key:value pairs  
    Assumes value is a numeric type  
    """  
  
    tVal = sum([v for (k,v) in xD.items()])  
    return tVal  
  
    # Alternative  
    # tVal = sum(xD.values())
```

► Answer 4 continued on next slide

► Go to Activity

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Bags: Definitions

Bags: Implementations

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Dictionaries

Answer 4 Total Exercise (contd)

► Answer 4 Total Exercise — sample use

```
Python3>>> aD  
{ 'not': 1, 'be': 2, 'to': 2, 'or': 1}  
Python3>>> totalValue(aD)  
6
```

► Go to Activity

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Bags: Definitions

Bags: Implementations

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Python dict

Modifiers

► Modifiers in the *M269 Companion*

```
Python3>>> aD = {'to': 2, 'be': 2
...             , 'or': 1, 'not': 1}
Python3>>> aD
{'not': 1, 'be': 2, 'to': 2, 'or': 1}
Python3>>> aD['dobe'] = 2
Python3>>> aD
{'not': 1, 'be': 2, 'dobe': 2, 'to': 2, 'or': 1}
Python3>>> aD['do'] = 1
Python3>>> aD
{'do': 1, 'to': 2, 'or': 1
 , 'not': 1, 'be': 2, 'dobe': 2}
Python3>>> aD['be'] = 3
Python3>>> aD
{'do': 1, 'to': 2, 'or': 1
 , 'not': 1, 'be': 3, 'dobe': 2}
```

Python dict

Modifiers (2)

► Modifiers in the *M269 Companion*

```
Python3>>> aD
{'do': 1, 'to': 2, 'or': 1
, 'not': 1, 'be': 3, 'dobe': 2}
Python3>>> aD['do'] = aD['do'] + 1
Python3>>> aD
{'do': 2, 'to': 2, 'or': 1
, 'not': 1, 'be': 3, 'dobe': 2}
Python3>>> del aD['dobe']
Python3>>> aD
{'do': 2, 'to': 2, 'or': 1
, 'not': 1, 'be': 3}
```

Python dict

Errors

- It is an error to access a non-existent key

```
Python3>>> aD
{'do': 2, 'to': 2, 'or': 1
, 'not': 1, 'be': 3}
Python3>>> aD['dobe']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'dobe'
Python3>>> del aD['dobe']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'dobe'
Python3>>> aD['dobe'] = aD['dobe'] + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'dobe'
```

Python dict

Errors (2)

- If a key occurs more than once, the last value for that key becomes the corresponding value in the new dictionary. Not an error but could catch you out

```
Python3>>> fD = {'one' : 2, 'one': 3}
Python3>>> fD
{'one': 3}
```

Dictionaries

Activity 5 Add One Exercise

- Write a function, `addOneToKey`, that takes a key, `key`, a dictionary, `xD`, and adds 1 to the value of the key

► Go to Answer

M269 Units 3,4,5

Phil Molyneux

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Sorting](#)

[Searching](#)

[String Search](#)

[Bags](#)

[Bags: Definitions](#)

[Bags: Implementations](#)

[Abstract Data Types](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic Programming](#)

[Future Work](#)

[Web Sites & References](#)

Dictionaries

Answer 5 Add One Exercise

► Answer 5 Add One Exercise

```
def addOneToKey(key, xD) :  
    """  
    addOneToKey takes a key and a dictionary  
    and adds one to the value of the key  
    Invariant for xD:  
        if key in xD :  
            xD[key] > 0  
    """  
  
    if key in xD :  
        xD[key] = xD[key] + 1  
    else :  
        xD[key] = 1  
  
    return xD
```

► Answer 5 continued on next slide

► Go to Activity

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Bags: Definitions

Bags: Implementations

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Dictionaries

Answer 5 Add One Exercise (contd)

- ▶ Answer 5 Add One Exercise
- ▶ Note that `addOneToKey` has a side effect on the input dictionary

```
Python3>>> aD
{'do': 2, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
Python3>>> addOneToKey('do', aD)
{'do': 3, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
Python3>>> aD
{'do': 3, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
Python3>>> addOneToKey('abba', aD)
{'do': 3, 'abba': 1, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
Python3>>> aD
{'do': 3, 'abba': 1, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
```

▶ Go to Activity

Dictionaries

Activity 6 Delete One Exercise

- Write a function, `delOneFromKey`, that takes a key, `key`, a dictionary, `xD`, and subtracts 1 from the value of the key if the key is in `xD`

► [Go to Answer](#)

M269 Units 3,4,5

Phil Molyneux

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Sorting](#)

[Searching](#)

[String Search](#)

[Bags](#)

[Bags: Definitions](#)

[Bags: Implementations](#)

[Abstract Data Types](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic Programming](#)

[Future Work](#)

[Web Sites & References](#)

Delete One Exercise

Answer 6 Delete One Exercise

► Answer 6 Delete One Exercise

```
def delOneFromKey(key, xD) :  
    """  
    delOneFromKey takes a key and a dictionary  
    and deletes one from the value of the key  
    Invariant for xD:  
        if key in xD :  
            xD[key] > 0  
    """  
  
    if not key in xD :  
        pass  
    elif xD[key] == 1 :  
        del xD[key]  
    else :  
        xD[key] = xD[key] - 1  
  
    return xD
```

► Answer 6 continued on next slide

Delete One Exercise

Answer 6 Delete One Exercise (contd)

► Answer 6 Delete One Exercise — examples

```
Python3>>> aD
{'do': 3, 'abba': 1, 'to': 2, 'or': 1
, 'not': 1, 'be': 3}
Python3>>> delOneFromKey('do',aD)
{'do': 2, 'abba': 1, 'to': 2, 'or': 1
, 'not': 1, 'be': 3}
Python3>>> aD
{'do': 2, 'abba': 1, 'to': 2, 'or': 1
, 'not': 1, 'be': 3}
Python3>>> delOneFromKey('abba',aD)
{'do': 2, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
Python3>>> aD
{'do': 2, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
Python3>>> delOneFromKey('bbc',aD)
{'do': 2, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
Python3>>> aD
{'do': 2, 'to': 2, 'or': 1, 'not': 1, 'be': 3}
```

► Go to Activity

Abstract Data Types

Overview

- ▶ **Abstract data type** is a type with associated operations, but whose representation is hidden (or not accessible)
- ▶ Common examples in most programming languages are Integer and Characters and other built in types such as tuples and lists
- ▶ **Abstract data types** are modeled on **Algebraic structures**
 - ▶ A set of values
 - ▶ Collection of operations on the values
 - ▶ Axioms for the operations may be specified as equations or pre and post conditions
- ▶ **Health Warning** different languages provide different ways of doing data abstraction with similar names that may mean subtly different things

Abstract Data Types

Overview (2)

- ▶ **Abstract Data Types and Object-Oriented Programming**
- ▶ Example: **Shape** with **Circles**, **Squares**, ... and operations **draw**, **moveTo**, ...
- ▶ **ADT** approach centres on the data type — that tells you what shapes exist
- ▶ For each operation on shapes, you describe what they do for different shapes.
- ▶ **OO** you declare that to be a shape, you have to have some operations (**draw**, **moveTo**)
- ▶ For each kind of shape you provide an implementation of the operations
- ▶ **OO** easier to answer *What is a circle?* and add new shapes
- ▶ **ADT** easier to answer *How do you draw a shape?* and add new operations

Abstract Data Types

Overview (3)

- ▶ **Health Warning and Optional Material** Discussions about the merits of [Functional programming](#) and [Object-oriented programming](#) tend to look like the disputes between [Lilliput and Blefuscu](#)
- ▶ [Abstract data type](#) article contrasts ADT and OO as algebra compared to co-algebra
- ▶ [What does *coalgebra* mean in the context of programming?](#) is a fairly technical but accessible article.
- ▶ [What does the *forall* keyword in Haskell do?](#) — is an accessible article on *Existential Quantification*
- ▶ [Bart Jacobs Coalgebra](#)
- ▶ [nLab Coalgebra](#)
- ▶ Beware the distinction between *concepts* and *features* in programming languages — see [OOP Disaster](#)
- ▶ **Not for this session** — this slide is here just in case

Abstract Data Types

Overview (4) — Shapes ADT Style

```
1 data Shape
2   = Circle Point Radius
3   | Square Point Size

5 draw :: Shape -> Pict
6 draw (Circle p r) = drawCircle p r
7 draw (Square p s) = drawRectangle p s s

9 moveTo :: Point -> Shape -> Shape
10 moveTo p2 (Circle p1 r) = Circle p2 r
11 moveTo p2 (Square p1 s) = Square p2 s

13 shapes :: [Shape]
14 shapes = [Circle (0,0) 1, Square (1,1) 2]

16 shapes01 :: [Shape]
17 shapes01 = map (moveTo (2,2)) shapes
```

- Example based on Lennart Augustsson email of 23 June 2005 on Haskell list

Abstract Data Types

Overview (5) — Shapes OO Style

```
1 class IsShape shape where
2   draw :: shape -> Pict
3   moveTo :: Point -> shape -> shape

5 data Shape = forall a . (IsShape a) => Shape a

7 data Circle = Circle Point Radius
8 instance IsShape Circle where
9   draw (Circle p r) = drawCircle p r
10  moveTo p2 (Circle p1 r) = Circle p2 r

12 data Square = Square Point Size
13 instance IsShape Square where
14   draw (Square p s) = drawRectangle p s s
15   moveTo p2 (Square p1 s) = Square p2 s

17 shapes :: [Shape]
18 shapes = [Shape (Circle (0,0) 10), Shape (Square (1,1) 2)]

20 shapes01 :: [Shape]
21 shapes01 = map (moveShapeTo (2,2)) shapes
22           where
23             moveShapeTo p (Shape s) = Shape (moveTo p s)
```


Abstract Data Types

Overview (6)

- ▶ Haskell Type Classes are similar to Java/OOP Interfaces
- ▶ See [OOP vs type classes](#)
- ▶ See [Java's Interface and Haskell's type class: differences and similarities?](#)
- ▶ See [Difference between OOP interfaces and FP type classes](#)
- ▶ See [What exactly makes the Haskell type system so revered \(vs say, Java\)?](#)
- ▶ **Health Warning** Much of OO programming is using the *OO syntax* to create ADTs

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Sorting](#)

[Searching](#)

[String Search](#)

[Bags](#)

[Abstract Data Types](#)

[Abstract Data Types — Overview](#)

[Marathon Example](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic Programming](#)

[Future Work](#)

[Web Sites & References](#)

Abstract Data Types

Marathon Example — Description

- ▶ Marathon example from M269 2017J TMA02 Q 2 — description
- ▶ **Creator**
- ▶ `mthon = Marathon()` sets up a new marathon
- ▶ **Modifiers**
- ▶ `mthon.register(runner)` adds *runner* (unique string) to the marathon — called before race
- ▶ `mthon.finish(runner, time)` records finishing time (seconds) of *runner*. Rounding means several runners may have same time. Only called in the race via sensor on finish mat

M269 Units 3,4,5

Phil Molyneux

[Agenda](#)

[Adobe Connect](#)

[Recursion](#)

[Sorting](#)

[Searching](#)

[String Search](#)

[Bags](#)

[Abstract Data Types](#)

[Abstract Data Types — Overview](#)

[Marathon Example](#)

[Marathon Example Questions](#)
[Code](#)

[Search Trees](#)

[Graph Algorithms](#)

[Greedy Algorithms](#)

[Dynamic Programming](#)

[Future Work](#)

[Web Sites & References](#)

Abstract Data Types

Marathon Example — Description (2)

- ▶ **Inspectors** only called after the race
- ▶ `mthon.place(runner)` returns place of the runner — places start at 1 (not 0) for shortest time
- ▶ `mthon.name(place)` returns the username of the runner in the given place.
- ▶ **Inspectors** called at any time
- ▶ `mthon.registered(runner)` returns `True` only if runner is registered
- ▶ `mthon.finished(runner)` returns `True` only if runner has finished the race
- ▶ `mthon.finishers()` returns the number of runners who have finished so far
- ▶ `mthon.finishersUpTo(time)` returns number of runners finished so far in given time (seconds) or less.

Marathon Example

Questions

1. Describe how to assign different places to runners, even those with the same finishing times
2. State the preconditions for all functions excepts for `Marathon()`, `finishers()` and `finishersUpTo(time)`
3. Justify the data structures used and mention any alternatives considered
4. Implement the ADT
5. State the worst case complexity of each operation

Marathon Example

Question: Assign places

- Which of the following are feasible patterns of **finish** calls ?

A.

```
mthon.finish('alice',180*60)
mthon.finish('bob',165*60)
mthon.finish('chris',185*60)
```

B.

```
mthon.finish('alice',180*60)
mthon.finish('bob',185*60)
mthon.finish('alice',185*60)
```

C.

```
mthon.finish('alice',180*60)
mthon.finish('bob',185*60)
mthon.finish('chris',185*60)
```

D.

```
mthon.finish('alice',180*60)
mthon.finish('chris',185*60)
mthon.finish('bob',185*60)
```

Marathon Example

Question: Assign places

- ▶ Feasible patterns of **finish** calls ?
- ▶ Not feasible A (wrong order), B (duplicate entry)
- ▶ Feasible C, D (but *bob* and *chris* get different places)
- ▶ Hence assume the n^{th} call to **finish** is the n^{th} runner to finish — so same time is ordered by sequence of **finish** calls
- ▶ The function **finish** will allocate place n and the time in the *Marathon* data structure

Marathon Example

Question: Preconditions

- ▶ What should the preconditions to **finish** be ?
- ▶ Investigate preconditions by writing down the *type* of **finish**
- ▶ In pseudo-code (Haskell)

```
finish :: Marathon -> (Runner, Time) -> Marathon
```

- ▶ In words — **finish** takes a **Marathon** and a pair of **Runner** (String) and **Time** (Int minutes) and returns a **Marathon** (updated)
- ▶ Note we are abusing Haskell notation here since **finish** has a side effect on a marathon

Marathon Example

Types of Functions

- ▶ Write down the *types* of the remaining functions of the Marathon
- ▶ register
- ▶ finish
- ▶ place
- ▶ name
- ▶ registered
- ▶ finished
- ▶ finishers
- ▶ finishersUpTo

Marathon Example

Types of Functions

► Types of functions

```
register :: Marathon -> Runner -> Marathon
```

```
finish :: Marathon -> (Runner, Time) -> Marathon
```

```
place :: Marathon -> Runner -> Place
```

```
name :: Marathon -> Place -> Runner
```

```
registered :: Marathon -> Runner -> Bool
```

```
finished :: Marathon -> Runner -> Bool
```

```
finishers :: Marathon -> Int
```

```
finishersUpTo :: Marathon -> Time -> Int
```

Marathon Example

Preconditions and Marathon Data Structure

- ▶ **Discussion** We have to choose our representation of **Marathon** and write the `__init__(self)` method
- ▶ **Lists** are mutable **sequences**
- ▶ **Tuples** are immutable **sequences**
- ▶ **Dictionaries** are mutable objects that can look up values by key (hashable name) such as a string
- ▶ With lists and tuples you can get out of range errors
- ▶ With dictionaries it is an error to look up a key that is not in the map
- ▶ Use the types to work out how we want to access the data and what choices we have to store the data
- ▶ What is the data type of **Runner** ?
- ▶ What is the data type of **Place** ?

Marathon Example

Preconditions and Marathon Data Structure

Operation	Preconditions
register	not registered(<i>runner</i>)
finish	registered(<i>runner</i>) and <i>time</i> > 0
place	finished(<i>runner</i>)
name	$0 < place \leq finishers()$
registered	True
finished	registered(<i>runner</i>)

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Abstract Data Types —
Overview

Marathon Example

Marathon Example
Questions

Code

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Marathon Example

Representation and Code

```
12 class Marathon :
13     # Creator
14     # -----
15
16     def __init__(self) :
17         """
18         Set up an empty Marathon
19         """
20         self.runnerToPlace = dict()
21         self.listOfRunnerTime = []
```

- ▶ **runnerToPlace** is a dictionary mapping *runner* to *place*
- ▶ Registered runners who have not finished have a place of 0
- ▶ **listOfRunnerTime** is a list of (*runner*, *time*) which maps *place* (list index + 1) to *runner* and *time*

Representation and Code

Modifier: register

```
81 # This modifier is called before the race starts
82 def register(self, runner) :
83     """
84     Register the runner
85     Returns nothing
86     Assumes the runner is not yet registered
87     """
88     self.runnerToPlace[runner] = 0
```

- **runnerToPlace** stores both *place* and whether registered

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Abstract Data Types —
Overview

Marathon Example

Marathon Example
Questions

Code

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Representation and Code

Modifier: finish

```
90 # This modifier is called during the race
91 def finish(self, runner, time) :
92     """
93     Record that the runner finished in given time
94     Return nothing
95     Assume the unit of time is seconds
96     Assume the runner has registered
97     """
98     place = self.finishers() + 1
99     self.listOfRunnerTime.append((runner, time))
100    self.runnerToPlace[runner] = place
```

- ▶ `listOfRunnerTime` gives the *place* by adding 1 to the list index
- ▶ This is equivalent to adding 1 to the number of runners who have finished so far

Representation and Code

Inspectors: registered, finishers

```
29 def registered(self, runner) :
30     """
31     Return True if runner has registered,
32     otherwise False
33     """
34     return runner in self.runnerToPlace

36 def finishers(self) :
37     """
38     Return number of finishers so far
39     """
40     return len(self.listOfRunnerTime)
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Abstract Data Types —
Overview

Marathon Example

Marathon Example
Questions

Code

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Representation and Code

Inspectors: finished, finishersUpTo

```
42 def finished(self, runner) :  
43     """  
44     Return True if runner has finished  
45     otherwise False  
46     Assume the runner has registered  
47     """  
48     return self.runnerToPlace[runner] > 0
```

```
52 def finishersUpTo(self, time) :  
53     """  
54     Return how many runners finished  
55     in given time or less  
56     Assume the unit of time is seconds  
57     """  
58     for index in range(self.finishers()) :  
59         if self.listOfRunnerTime[index][TIMEINDEX] > time :  
60             return index  
61     # All starting finished up to given time  
62     return self.finishers()
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Abstract Data Types —
Overview

Marathon Example

Marathon Example
Questions

Code

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Representation and Code

Inspectors: place, name

```
64 def place(self, runner) :  
65     """  
66     Return place the runner finished  
67     Assume runner finished  
68     """  
69     return self.runnerToPlace[runner]
```

```
71 def name(self, place) :  
72     """  
73     Return name of runner in given place  
74     Assume place corresponds to a finisher  
75     """  
76     return self.listOfRunnerTime[place - 1][NAMEINDEX]
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Abstract Data Types —
Overview

Marathon Example

Marathon Example
Questions

Code

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Search Trees

Height Balanced Trees

- ▶ See the slides, notes and examples in [M269 Tutorial Binary Trees](#)
- ▶ The section on AVL Trees gives the basic transformations
- ▶ The section on *Implementing Sets in AVL Trees* gives an application of AVL Trees

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Height Balanced Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Graph Algorithms

Using the M269 Library

- ▶ Download the [M269 Library](https://mwermelinger.github.io/m269-library/) from <https://mwermelinger.github.io/m269-library/>
- ▶ Copy the folders `lib`, `doc`, `examples`, `help`, `tests` to your working folder — I am using `UsingM269Library`
- ▶ `__init__.py`
- ▶ `bst.py`, `deque.py`, `digraph.py`, `graph.py`
- ▶ `hash_table.py`, `priority_queue.py`, `queue.py`, `sort.py`, `stack.py`
- ▶ Python Packages Documentation
- ▶ What is `__init__.py` for?
- ▶ Python Packaging Tutorials

Using the M269 Library

UsingM269Library.py

```
1#!/usr/bin/env python3
3from lib.graph import *
5# Sample M269 Library graphs
7pwrGr = Graph()
9pwrGr.add_edge(1, 2, 100)
10pwrGr.add_edge(2, 3, 100)
11pwrGr.add_edge(3, 4, 50)
12pwrGr.add_edge(3, 5, 50)
13pwrGr.add_edge(6, 2, 50)
14pwrGr.add_edge(6, 7, 50)
15pwrGr.add_edge(7, 8, 40)
16pwrGr.add_edge(8, 9, 10)
17pwrGr.add_edge(8, 10, 10)
18pwrGr.add_edge(8, 11, 20)
```

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Using the M269 Library

Comparisons in Python

The import Statement
and Modules

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Using the M269 Library

Using the Test Code

```
<UsingM269Library><108> python3
Python3>>> from UsingM269Library import *
Python3>>> pwrGr.nodes()
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
Python3>>> type(pwrGr.nodes())
<class 'set'>
Python3>>> pwrGr.visited_dfs(2)
[2, 6, 7, 8, 11, 10, 9, 3, 5, 4, 1]
Python3>>> type(pwrGr.visited_dfs(2))
<class 'list'>
Python3>>> pwrGr.visited_dfs(2) == pwrGr.nodes()
False
Python3>>> set(pwrGr.visited_dfs(2))
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
Python3>>> pwrGr.visited_dfs(2) == len(pwrGr.nodes())
False
Python3>>> len(pwrGr.visited_dfs(2)) == len(pwrGr.nodes())
True
Python3>>> set(pwrGr.visited_dfs(2)) == pwrGr.nodes()
True
Python3>>>
```

Comparisons in Python

Using M269 Library

- ▶ Notice in the above examples comparing a list with set returned `False`, and comparing a list with the size (`len`) of a set returned `False`
- ▶ If you had previously used strongly typed languages you might have expected a `TypeError`
- ▶ **Python Library: Comparisons** says: *Objects of different types, except different numeric types, never compare equal.*
- ▶ Function objects always compare unequal (even though comparing functions for equality is in general not computable — see Unit 7)
- ▶ The last two examples give correct ways of comparing — though it would be best to do set equality comparison in this case.
- ▶ This must be a big source of bugs (and a shock to people like me)

Comparisons in Python

Rationale for Equality Comparison across Types

- ▶ Python is weakly typed
- ▶ Dictionaries can have keys of mixed types — for example, integer `1` and string `'1'`
- ▶ See [Python: Why does equality comparing an int with a string not throw an error?](#)
- ▶ See [Python: dict](#)

Modules

import Statement

- ▶ Python Language: The `import` statement is the reference for the `import` statement

```
from lib.graph import *
```

- ▶ All public names in `graph.py` are bound in the local namespace and we can refer to `Graph()`

```
import lib.graph
```

- ▶ `lib.graph` imported and we have to refer to `lib.graph.Graph()`
- ▶ Python Tutorial: Modules are described in the Python Tutorial — notice section 6.4.1 *Importing * from a Package*
- ▶ Python Command Line: `PYTHONPATH` can augment the default search path for module files
- ▶ The Python search path can be manipulated with `sys.path`

Greedy Algorithms

Overview

- ▶ **Greedy algorithms** follow the problem solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum
- ▶ In general this rarely works — but it does in some cases including
 - ▶ **Dijkstra's algorithm** and **A* search algorithm** for graph search and shortest path finding
 - ▶ **Kruskal's algorithm** and **Prim's algorithm** for constructing minimum spanning trees of a given connected graph
 - ▶ **Interval scheduling** or **Activity selection problem** to find the maximum number of activities that do not clash with each other
- ▶ If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice because it is faster than other optimization methods such as **dynamic programming**.

Greedy Algorithms

Interval Scheduling

- ▶ **Interval scheduling**
- ▶ Job j starts at s_j and finishes at f_j
- ▶ Two jobs are compatible if they do not overlap
- ▶ **Q** What is the maximum subset of mutually compatible jobs?
- ▶ **Greedy template** Consider jobs in some order. Take each job provided it is compatible with the ones already taken.
- ▶ **Exercise** What orderings can we have ?
- ▶ Example from **Greedy algorithms: Interval scheduling**

Greedy Algorithms

Interval Scheduling

- ▶ **Greedy template** Consider jobs in some order. Take each job provided it is compatible with the ones already taken.
- ▶ **Earliest start time** Consider jobs in ascending order of start time s_j
- ▶ **Earliest finish time** Consider jobs in ascending order of finish time f_j
- ▶ **Shortest interval** Consider jobs in ascending order of interval length $f_j + 1 - s_j$
- ▶ **Fewest conflicts** For each job, count the number of conflicting jobs c_j and schedule in ascending order of conflicts c_j

Interval Scheduling

Example

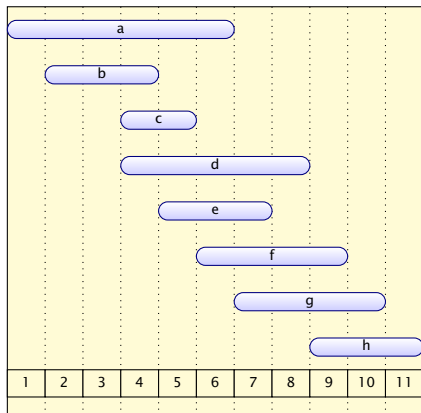
- ▶ For the jobs given below, produce an ordering by each of the greedy templates (above) and the schedule produced
- ▶ Each triple in the list below means $(name, s_i, f_i)$ where the times are inclusive

```
jobs  
= [(a,1,6), (b,2,4), (c,4,5), (d,4,8)  
  , (e,5,7), (f,6,9), (g,7,10), (h,9,11)]
```

[Agenda](#)[Adobe Connect](#)[Recursion](#)[Sorting](#)[Searching](#)[String Search](#)[Bags](#)[Abstract Data Types](#)[Search Trees](#)[Graph Algorithms](#)[Greedy Algorithms](#)[Interval Scheduling](#)[Dynamic Programming](#)[Future Work](#)[Web Sites & References](#)

Interval Scheduling

Order by Earliest Start Time



► Schedule jobs **a**, **g** (2 jobs)

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Interval Scheduling

Dynamic

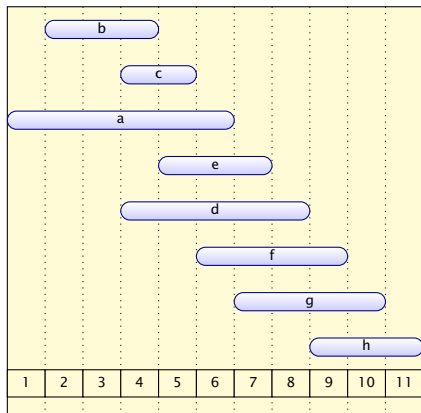
Programming

Future Work

Web Sites &
References

Interval Scheduling

Order by Earliest Finish Time



► Schedule jobs **b, e, h** (3 jobs)

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Interval Scheduling

Dynamic

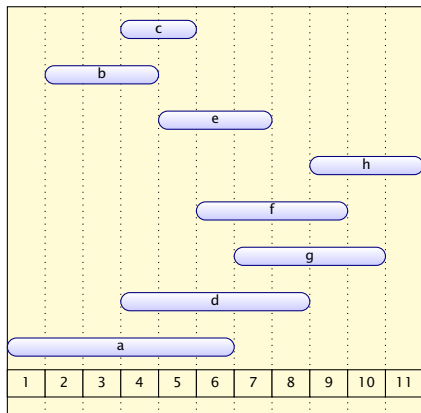
Programming

Future Work

Web Sites &
References

Interval Scheduling

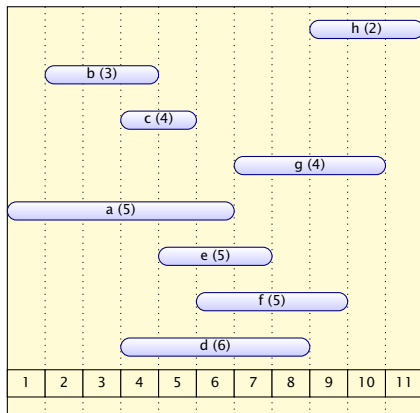
Order by Shortest Interval



► Schedule jobs **c, h** (2 jobs)

Interval Scheduling

Order by Fewest Conflicts



► Schedule jobs **h**, **b**, **e** (3 jobs)

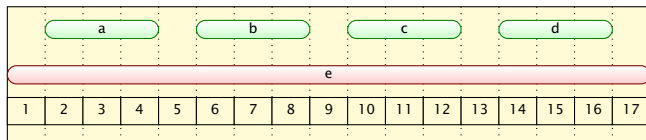
Interval Scheduling

Counter Examples

- ▶ For each of the following *Greedy Templates* produce a counter example to show it may not produce the optimal schedule
- ▶ **Earliest start time** Consider jobs in ascending order of start time s_j
- ▶ **Shortest interval** Consider jobs in ascending order of interval length $f_j + 1 - s_j$
- ▶ **Fewest conflicts** For each job, count the number of conflicting jobs c_j and schedule in ascending order of conflicts c_j

Interval Scheduling

Order by Earliest Start Time — Counter Example



- **e** dominates the optimal schedule by starting earlier and overlapping the others

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Interval Scheduling

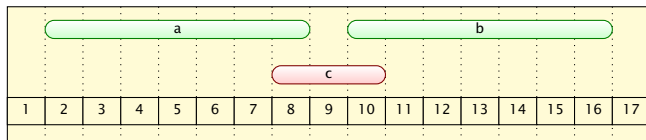
Dynamic Programming

Future Work

Web Sites & References

Interval Scheduling

Order by Shortest Interval — Counter Example

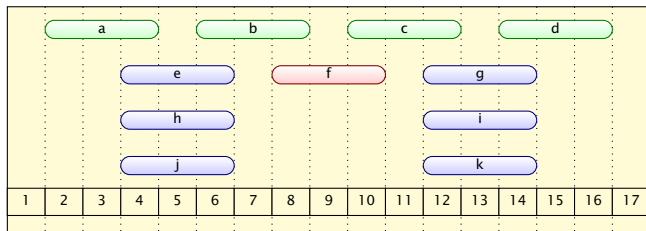


- **c** dominates the optimal schedule **y** being shorter and overlapping the other two

[Agenda](#)[Adobe Connect](#)[Recursion](#)[Sorting](#)[Searching](#)[String Search](#)[Bags](#)[Abstract Data Types](#)[Search Trees](#)[Graph Algorithms](#)[Greedy Algorithms](#)[Interval Scheduling](#)[Dynamic Programming](#)[Future Work](#)[Web Sites & References](#)

Interval Scheduling

Order by Fewest Conflicts — Counter Example



- **f** dominates the optimal schedule by only having two conflicts and overlapping **b** and **c**

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Interval Scheduling

Dynamic Programming

Future Work

Web Sites & References

Interval Scheduling

Order by Earliest Finish Time — Optimality Proof

- ▶ Basic structure of correctness proof:
- ▶ Assume that there is an optimal solution that is different from the greedy solution.
- ▶ Find the *first* difference between the two solutions.
- ▶ Argue that we can exchange the optimal choice for the greedy choice without making the solution worse (although the exchange might not make it better).
- ▶ This argument implies by induction that some optimal solution *contains* the entire greedy solution, and therefore *equals* the greedy solution.
- ▶ Sometimes, an additional step is required to show no optimal solution *strictly* improves the greedy solution.
- ▶ See [Jeff Erickson: Algorithms](#)
- ▶ Proof also in [Interval Scheduling](#) and [Greedy Algorithms](#)
- ▶ The slides at [Kevin Wayne: Greedy Algorithms](#) are from Kleinberg (2013)

[Agenda](#)[Adobe Connect](#)[Recursion](#)[Sorting](#)[Searching](#)[String Search](#)[Bags](#)[Abstract Data Types](#)[Search Trees](#)[Graph Algorithms](#)[Greedy Algorithms](#)[Interval Scheduling](#)[Dynamic Programming](#)[Future Work](#)[Web Sites & References](#)

Dynamic Programming

Edit Distance

- ▶ See the slides, notes and examples in [M269 Tutorial Graph Algorithms](#)
- ▶ The section on Dynamic Programming gives the basic formulation
- ▶ The section on *Edit Distance* gives an application

Future Work

Topics & Events

- ▶ Wednesday, 24 March 2021 TMA02 due
- ▶ Sunday, 11 April 2021 online tutorial Unit 6 Logic
- ▶ Wednesday 28 April 2021 iCMA46 due
- ▶ Sunday, 3 May 2020 online tutorial Unit 7
Computability, Complexity
- ▶ Sunday, 16 May 2021 online tutorial exam revision
- ▶ Saturday, 22 May 2021 tutorial online exam revision
- ▶ Tuesday 25 May 2021 iCMA47 due
- ▶ Tuesday 8 June 2021 Exam
- ▶ Please email me with any requests for particular topics

Web Sites

Sorting

- **Big-O Cheat Sheet** <http://bigocheatsheet.com/>

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Sorting

Graph Algorithms

Web Sites

Graph Algorithms

- ▶ **Graph Theory** http://en.wikipedia.org/wiki/Graph_theory
- ▶ **Tree (Graph Theory)** [http://en.wikipedia.org/wiki/Tree_\(graph_theory\)](http://en.wikipedia.org/wiki/Tree_(graph_theory))
- ▶ **Dekai Wu Algorithms course** <https://www.cse.ust.hk/~dekai/271/>
- ▶ **Jeff Erickson Algorithms** <http://jeffe.cs.illinois.edu/teaching/algorithms/>

M269 Units 3,4,5

Phil Molyneux

Agenda

Adobe Connect

Recursion

Sorting

Searching

String Search

Bags

Abstract Data Types

Search Trees

Graph Algorithms

Greedy Algorithms

Dynamic
Programming

Future Work

Web Sites &
References

Sorting

Graph Algorithms

Web Sites

Dynamic Programming

- ▶ *Jeff Erickson Algorithms*
jeffe.cs.illinois.edu/teaching/algorithms
- ▶ *Cormen* (2009, page 407) has same example as *Jeff Erickson*
- ▶ *Peter Norvig Spell Checker*
norvig.com/spell-correct.html (from
stackoverflow.com/questions/2460177/edit-distance-in-python)
- ▶ *Rosetta Code: Levenshtein Distance*
rosettacode.org/wiki/Levenshtein_distance
- ▶ *Edit Distance HaskellWiki*
https://wiki.haskell.org/Edit_distance
- ▶ *Edit Distance in Haskell*
stackoverflow.com/questions/5515025/edit-distance-algorithm-in-haskell-performance-tuning
- ▶ *Wikibooks Algorithm implementation — Levenshtein Distance* [en.wikibooks.org/wiki/Algorithm_-Implementation/Strings/Levenshtein_distance](http://en.wikibooks.org/wiki/Algorithm_-_Implementation/Strings/Levenshtein_distance)
- ▶ *Wikipedia Levenshtein Distance*
en.wikipedia.org/wiki/Levenshtein_distance