M269 Python, Logic,ADTs

M269 Python, ADTs Prsntn2024J

Phil Molyneux

24 November 2024

M269 Tutorial: Python, Logic, ADTs

Agenda

- Introductions
- Programming — Paradigms and Step-by-Step Guide
- Programming and Python
- Complexity and Big-O/Big-Theta Notation
- ...with a little classical logic
- Abstract Data Type examples
- Implementing Lists in Lists
- A look towards the next topics
  - Beware: some topics are are included for interest only and are not part of M269
  - These notes use recursion, explained at the time
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)
- Time: about 1 hour
- Do ask questions or raise points.
- Slides/Notes M269Tutorial20241124ProgPythonADTPrsntn2024J

M269 Tutorial

Introductions — Phil

- *Name* Phil Molyneux
- *Background*
  - Undergraduate: Physics and Maths (Sussex)
  - Postgraduate: Physics (Sussex), Operational Research (Brunel), Computer Science (University College, London)
  - Worked in Operational Research, Business IT, Web technologies, Functional Programming
- *First programming languages* Fortran, BASIC, Pascal
- *Favourite Software*
  - Haskell — pure functional programming language
  - Text editors TextMate, Sublime Text — previously Emacs
  - Word processing in LaTeX — all these slides and notes
  - Mac OS X
- *Learning style* — I read the manual before using the software

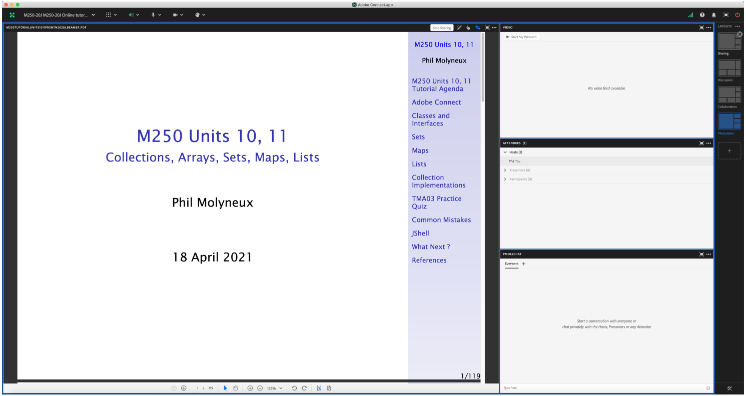
M250 Tutorial

Introductions — You

- *Name* ?
- *Favourite software/Programming language* ?
- *Favourite text editor or integrated development environment (IDE)*
- List of text editors, Comparison of text editors and Comparison of integrated development environments
- *Other OU courses* ?
- *Anything else* ?

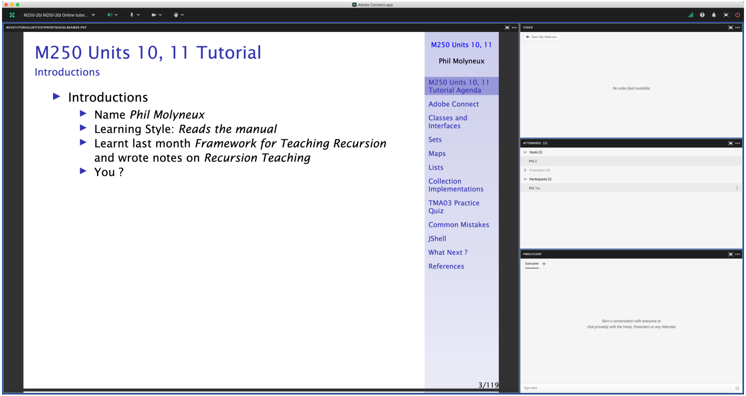
Adobe Connect

Interface — Host View



Adobe Connect

Interface — Participant View



Adobe Connect

Settings

- **Everybody** *Menu bar* › *Meeting* › Speaker & Microphone Setup
- *Menu bar* › *Microphone* › Allow Participants to Use Microphone ✔
- Check Participants see the entire slide **Workaround**
  - *Disable Draw* *Share pod* › *Menu bar* › *Draw icon*
  - *Fit Width* *Share pod* › *Bottom bar* › *Fit Width icon* ✔
- *Meeting* › Preferences › General › **Host Cursor** › Show to all attendees
- *Menu bar* › *Video* › Enable **Webcam** for Participants ✔
- Do not *Enable single speaker mode*
- Cancel hand tool
- Do not enable green pointer
- **Recording** Meeting › Record Session ✔
- **Documents** Upload PDF with drag and drop to share pod
- Delete *Meeting* › Manage Meeting Information › Uploaded Content and *check filename* › *click on delete*

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

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** ⌘ + \

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 blued 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

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.
- **Delete File Upload** Meeting › Manage Meeting Information › Uploaded Content tab select file(s) and click Delete
- **Attendance Report** see course Web site for joining room

Adobe Connect

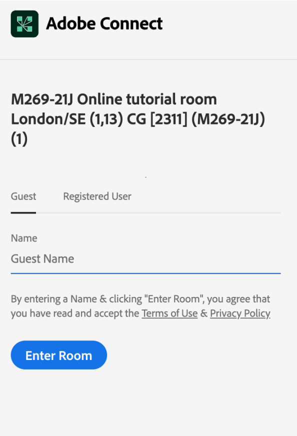
Invite Attendees

- **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

Adobe Connect

Entering a Room as a Guest (1)

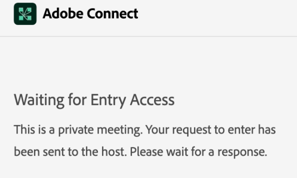
- Click on the link sent in email from the Host
- Get the following on a Web page
- As *Guest* enter your name and click on **Enter Room**



Adobe Connect

Entering a Room as a Guest (2)

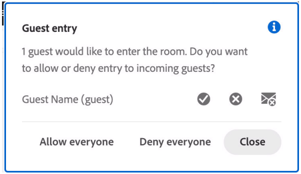
- See the *Waiting for Entry Access* for *Host* to give permission



Adobe Connect

Entering a Room as a Guest (3)

- *Host* sees the following dialog in *Adobe Connect* and grants access



Adobe Connect

Layouts

- **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 *PMolyShare*
- **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 — rename it *PMolyChat* — and resize/reposition

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)
- **Auxiliary Layouts** name *PMolyAux0n*
  - Create new Share pod
  - Use existing Chat pod
  - Use same Video and Attendance pods

Adobe Connect

Chat Pods

- **Format Chat text**
- Chat Pod > menu icon > My Chat Color
- Choices: Red, Orange, Green, Brown, Purple, Pink, Blue, Black
- Note: Color reverts to Black if you switch layouts
- Chat Pod > menu icon > Show Timestamps

Graphics Conversion

PDF to PNG/JPG

- Conversion of the screen snaps for the installation of Anaconda on 1 May 2020
- Using GraphicConverter 11
- File > Convert & Modify > Conversion > Convert
- Select files to convert and destination folder
- Click on Start selected Function or ⌘ + ⏎

Adobe Connect Recordings

Exporting Recordings

- *Menu bar* > *Meeting* > Preferences > Video
- Aspect ratio > Standard (4:3) (not Wide screen (16:9) default)
- Video quality > Full HD (1080p not High default 480p)
- **Recording** *Menu bar* > *Meeting* > Record Session ✔
- **Export Recording**
- *Menu bar* > *Meeting* > Manage Meeting Information
- *New window* > Recordings > *check Tutorial* > Access Type button
- check Public > check Allow viewers to download
- **Download Recording**
- *New window* > Recordings > *check Tutorial* > Actions > Download File

Computational Components

Imperative, Procedural Programming

Imperative or procedural programming has statements which can manipulate global memory, have explicit control flow and can be organised into procedures (or functions)

- **Sequence** of statements

```
stmnt ; stmnt
```

- **Iteration** to repeat statements

```
while expr :
  suite

for targetList in exprList :
  suite
```

- **Selection** choosing between statements

```
if expr : suite
elif expr : suite
else : suite
```

Computational Components

Functional Programming

Functional programming treats computation as the evaluation of expressions and the definition of functions (in the mathematical sense)

- **Function composition** to combine the application of two or more functions — like sequence but from right to left (notation accident of history)

```
(f . g) x = f (g x)
```

- **Recursion** — function definition defined in terms of calls to itself (with *smaller* arguments) and base case(s) which do not call itself.
- **Conditional expressions** choosing between alternatives expressions

```
if expr then expr else expr
```

Computation

Programming, Programming Languages

- M269 is not a programming course but ...
- The course uses Python to illustrate various algorithms and data structures
- The final unit addresses the question:
- *What is an algorithm* ? What is programming ? What is a programming language ?
- So it *is* a programming course (sort of)

Example Algorithm Design

Searching

- 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)

Example Algorithm Design

Binary Search Iterative

```python
def binarySearchIter(xs,val):
  lo = 0
  hi = len(xs) - 1

  while lo <= hi:
    mid = (lo + hi) // 2
    guess = xs[mid]

    if val == guess:
      return mid
    elif val < guess:
      hi = mid - 1
    else:
      lo = mid + 1

  return None
```

Example Algorithm Design

Binary Search Recursive

```python
def binarySearchRec(xs,val,lo=0,hi=-1):
  if (hi == -1):
    hi = len(xs) - 1

  mid = (lo + hi) // 2

  if hi < lo:
    return None
  else:
    guess = xs[mid]
    if val == guess:
      return mid
    elif val < guess:
      return binarySearchRec(xs,val,lo,mid-1)
    else:
      return binarySearchRec(xs,val,mid+1,hi)
```

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 31
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 31
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: ??
```

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

```
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 31
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 31
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 29
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 31
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 31
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 29
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 31
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 31
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 29
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,7) by line 29
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 31
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,10) by line 31
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,8) by line 29
xs = [2,5,7,15,17,19,21,24,27,31,37,48,57,87,95]
binarySearchRec(xs,25,8,7) by line 29
Return value: None by line 23
```

Example Algorithm Design

Binary Search — Comparison

- 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 ?

Example Algorithm Design

Binary Search — Performance

- *Linear search* — number of comparisons
  - *Best case* 1 (first item in the list)
  - *Worst case* n (last item)
  - *Average case* $\frac{1}{2}n$
- *Binary search* — number of comparisons
  - *Best case* 1 (middle item in the list)
  - *Worst case* $\log_2 n$ (steps to see all)
  - *Average case* $\log_2 n - 1$ (steps to see half)

Writing Programs & Thinking

The Steps

1. Invent a *name* for the program (or function)
2. What is the *type* of the function ? What sort of *input* does it take and what sort of *output* does it produce ? In Python a type is implicit; in other languages such as Haskell a type signature can be explicit.
3. Invent *names* for the input(s) to the function (*formal parameters*) — this can involve thinking about possible *patterns* or *data structures*
4. What restrictions are there on the input — state the preconditions.
5. What must be true of the output — state the postconditions.
6. *Think* of the definition of the function body.

Writing Programs & Thinking

The *Think* Step

- **How to Think**
  1. Think of an example or two — what should the program/function do ?
  2. Break the inputs into separate cases.
  3. Deal with simple cases.
  4. Think about the result — try your examples again.
- **Thinking Strategies**
  1. Don't think too much at one go — break the problem down. Top down design, step-wise refinement.
  2. What are the inputs — describe all the cases.
  3. Investigate choices. What data structures ? What algorithms ?
  4. Use common tools — bottom up synthesis.
  5. Spot common function application patterns — generalise & then specialise.
  6. Look for good *glue* — to combine functions together.

Python

Learning Python

- Python 3 Documentation
- Python Tutorial
- Python Language Reference
- Python Library Reference
- Hitchhiker's Guide to Python
- Stackoverflow on Python
- Martelli et al (2023) *Python in a Nutshell*
- Lutz (2025) *Learning Python*

Basic Python

Python Usage — Questions

- How do you enter an interactive Python shell ?
- How do you exit Python in *Terminal* (Mac) or *Command prompt* (Windows) ?
- How do you get help in a shell ?
- How do you exit the *interactive help utility* ?

Basic Python

Python Usage — Answers

- How do you enter an interactive Python shell ?
- How do you exit Python in *Terminal* (Mac) or *Command prompt* (Windows) ?
- How do you get help in a shell ?
- How do you exit the *interactive help utility* ?

Basic Python

Python Usage — Answers

- How do you enter an interactive Python shell ?

  Windows PythonWin Shell from *Toolbox*; *Mac python3* in *Terminal*
- How do you exit Python in *Terminal* (Mac) or *Command prompt* (Windows) ?
- How do you get help in a shell ?
- How do you exit the *interactive help utility* ?

Basic Python

Python Usage — Answers

- How do you enter an interactive Python shell ?

  Windows PythonWin Shell from *Toolbox*; *Mac python3* in *Terminal*

- How do you exit Python in *Terminal* (Mac) or *Command prompt* (Windows) ?

  `quit()`

- How do you get help in a shell ?

- How do you exit the *interactive help utility* ?

Basic Python

Python Usage — Answers

- How do you enter an interactive Python shell ?

  Windows PythonWin Shell from *Toolbox*; *Mac python3* in *Terminal*

- How do you exit Python in *Terminal* (Mac) or *Command prompt* (Windows) ?

  `quit()`

- How do you get help in a shell ?

  `help()`

- How do you exit the *interactive help utility* ?

Basic Python

Python Usage — Answers

- How do you enter an interactive Python shell ?

  Windows PythonWin Shell from *Toolbox*; *Mac python3* in *Terminal*

- How do you exit Python in *Terminal* (Mac) or *Command prompt* (Windows) ?

  `quit()`

- How do you get help in a shell ?

  `help()`

- How do you exit the *interactive help utility* ?

  `quit`

Basic Python

Sequences Indexing, Slices

- `xs[i:j:k]` is defined to be the sequence of items from index i to (j-1) with step k.
- If k is omitted or None, it is treated as `1`.
- If i or j are negative then they are relative to the end.
- If i is omitted or None use `0`.
- If j is omitted or None use `len(xs)`

Basic Python

Python Quiz — Lists

Given the following definitions

```
xs = [10.9,25,"Phil",3.14,42,1985]
ys = [[5]] * 3
```

Evaluate

```
xs[1]
xs[0]
xs[5]
ys
xs[1:3]
xs[::2]
xs[1:-1]
xs[-3]
xs[:]
ys[0].append(4)
```

Basic Python

Python Quiz — Lists — Answers

Given the following definitions

```
xs = [10.9,25,"Phil",3.14,42,1985]
ys = [[5]] * 3
```

Evaluate

```
xs[1]             == 25
xs[0]             == 10.9
xs[5]             == 1985
ys                == [[5],[5],[5]]
xs[1:3]           == [25, 'Phil']
xs[::2]           == [10.9, 'Phil', 42]
xs[1:-1]          == [25, 'Phil', 3.14, 42]
xs[-3]            == 3.14
xs[:]             == [10.9, 25, 'Phil', 3.14, 42, 1985]
ys[0].append(4)   == [[5, 4], [5, 4], [5, 4]]
```

Python Workflows

Komodo Python Workflow

1. Create *someProgram*.py with assignment statements defining variables and other data along with function definitions.
2. There may be auxiliary files with other definitions (for example, *Python Activity 2.2* has Stack.py with the *Stack* class definition) — this uses the *import* statement in *someProgram*.py

```python
from someOtherDefinitions import someIdentifier
```

3. Load *someProgram*.py into *Komodo Edit* and use the *Run Python File* macro from the *Toolbox*
4. For further results, edit the file in *Komodo Edit* and and use the *Save and Run* macro from the *Toolbox*

Python Workflows

Standalone Python Workflow

1. Create *someDefinitions*.py with assignment statements defining variables and function definitions.
2. In *Terminal* (Mac) or *Command Prompt* (Windows), navigate to *someDefinitions*.py and invoke the *Python 3* interpreter
3. Load *someDefinitions*.py into *Python 3* with one of

   ```
   from someDefinitions import *
   ```

   ```
   import someDefinitions as sdf
   ```

   The `as sdf` gives a shorter qualifier for the namespace — names in the file are now `sdf.x`
   Note that the commands are executed — any print statement will execute
4. At the *Python 3* interpreter prompt, evaluate expressions (may have side effects and alter definitions)

Python Workflows

Standalone Python Workflow 2

1. For further results, edit the file in *Your Favourite Editor* and use one of the following commands:

```python
reload(sdf)

import imp
imp.reload(sdf)
```

Note the use of the name sdf as opposed to the original name.
Read the following references about the dangers of reloading as compared to re-cycling *Python 3*

- How to re import an updated package while in Python Interpreter?
- How do I unload (reload) a Python module?
- Reloading Python modules
- How to dynamically import and reimport a file containing definition of a global variable which may change anytime

Python Checking Tools

Purpose & Installation

- M269 provides some Python checking tools: `ruff` and `allowed` — a description is in the M269 Book section 5.3.2 (these notes are based on Jason Clarke's notes)
- M269 software installation is documented at dsa-ou.github.io/m269-installer/
- `allowed` is documented at dsa-ou.github.io/allowed/ — it checks for permitted code
- `ruff` Web site is docs.astral.sh/ruff/ —- Ruff is an extremely fast Python linter and code formatter, written in Rust
- Both `allowed` and `ruff` are installed in the standard M269 24J software install — they are in the `venvs` folder, wherever that was installed (in my case in my home folder)
- The intention is that `allowed` checks you are only using Python contained in the M269 Book and `ruff` comments on Python style issues such as the extent to which your code complies with PEP 8 — Style Guide for Python Code

Python Code Checker

allowed Code Checker

- `allowed` uses a data file `m269-24j.json` to determine if the code has allowed features only
- JSON (JavaScript Object Notation) is a lightweight data-exchange format — effectively it is one up from CSV (Comma Separated Values)
- JSON has a Web site json.org/json-en.html which contains the definitive standard — JSON is not part of M269 and you do not need to read the documentation for the course but since JSON is so widely used you may be interested
- An appendix to Douglas Crockford's book has more on JSON (Crockford was one of the original movers behind JSON) — see Crockford (2008, Appendix E) *JavaScript: The Good Parts*

Python Code Checker

Using allowed

- Activate `allowed` (and `ruff`) by running the cell that appears early on in your Jupyter Notebook which contains some IPython Magic Commands (see examples below)
- When you run any Python cells, you will see any output from `allowed`and `ruff` as well as your own output
- If all your usage is allowed you will see no output from `allowed`
- If you have used something outside the permitted code you may see a message from `allowed`
- Note that the `allowed` tool is not perfect and there are some differences between Windows, macOS and Linux users (see examples below)
- All the allowed code is described or listed in the *Summary* sections at the end of each chapter in the M269 Book (or you could read the `m269-24j.json` file if you are very relaxed)

Python Code Checker

Activating allowed and ruff

- The TMA Jupyter Notebooks seem to have several versions of the software that activates `allowed` and `ruff` so this section of these notes may be subject to change
- From Section 5.3.2 of the M269 Book there are cells that has the following code

```
%load_ext algoesup.magics
%ruff on
```

```python
import platform  # allowed

if platform.system() in ('Linux', 'Darwin'):
    %allowed on --config m269-24j --unit 5 --method
else:
    %allowed on --config m269-24j --unit 5
```

Python Code Checker

Activating allowed and ruff

- TMA01 has the following

```
%load_ext algoesup.magics
%allowed on
%ruff on
%run -i m269_test
```

- The first version activates `allowed` for code in the first 5 chapters and methods are checked in Linux and macOS
- The second activates for code in all chapters

Python Code Checker

Allowed Methods

- **Allowed Methods** a *method* is of the form

  `objectName.methodName(args)`

- Examples

```
Python3>>> myList = [1,2,3]
Python3>>> myList.append(5) # using list append method
Python3>>> print(myList)
[1, 2, 3, 5]
Python3>>> myText = "abc"
Python3>>> myText.upper() # using string upper method
'ABC'
Python3>>> print(myText)
abc
```

- Some methods return values, others have side effects
- Read the Python documentation for the details

Python Code Checker

Allowed Methods

- From `m269-25j.json` we have the following allowed methods
- **List** insert, append, pop, sort
- **Dict** items, pop
- **Set** add, discard, union, intersection, difference, pop
- Note that **no string methods** are allowed

allowed Methods

Example Messages from Non-Allowed Methods

```
# List

myList = [1,2,3]

print(myList.count(2))
myList.extend([3,4])
myList.remove(2)
print(myList.index(1))
myList.reverse()
myList.clear()

1
0
```

allowed found issues

- 5: `list.count()`
- 6: `list.extend()`
- 7: `list.remove()`
- 8: `list.index()`
- 9: `list.reverse()`
- 10: `list.clear()`

allowed Methods

Example Messages from Non-Allowed Methods

```
# Anything on strings...

myText = "Hello"
print(myTextext.upper())
print(myText.find("e"))
print(myText.endswith("o"))
print(myText.isdigit())
sep = ","
print(sep.join(["A","B","C"]))

HELLO
1
True
False
A,B,C
```

allowed found issues

- 4: `str.upper()`
- 5: `str.find()`
- 6: `str.endswith()`
- 7: `str.isdigit()`
- 9: `str.join()`

allowed Methods

Non-Allowed Methods which Pass

- The above examples may not produce error messages on Windows platforms
- Expressions with literal object or literal arguments may be not-allowed but may not generate error messages
- Missing type hints for function definitions may result in non-allowed being missed
- The foollowing are non-allowed but produce no messages

```
print([1,2,3].index(1))  # method on a literal

print("abc".upper()) # method on a literal
print(",".join(["A","B","C"])) # literal argument

def test(txt):
  return txt.isdigit()  # Missing type hint cso missed

def anotherTest(txt : str):
  print(txt[0].upper()) # type hint - argument is an expression

0
ABC
A,B,C
```

allowed Methods

User Defined Methods

- **User Defined Methods**
- `allowed` will not generate error messages to these
- So you can write methods (or functions) to replace the the functionality of some other methods

Program Complexity

Big O Notation

- Measuring program complexity introduced in section 4 of M269 Unit 2
- See also Miller and Ranum chapter 2 Big-O Notation
- See also Wikipedia: Big O notation
- See also Big-O Cheat Sheet

Program Complexity

Big O Notation (2)

- Complexity of algorithm measured by using some surrogate to get rough idea
- In M269 mainly using assignment statements
- For *exact* measure we would have to have cost of each operation, knowledge of the implementation of the programming language and the operating system it runs under.
- But mainly interested in the following questions:
- **(1)** Is algorithm A more efficient than algorithm B for large inputs ?
- **(2)** Is there a lower bound on any possible algorithm for calculating this particular function ?
- **(3)** Is it always possible to find a polynomial time (n^k) algorithm for any function that is computable
- — the later questions are addressed in Unit 7

Program Complexity

Orders of Common Functions

- $O(1)$ **constant** — look-up table
- $O(\log n)$ **logarithmic** — binary search of sorted array, binary search tree, binomial heap operations
- $O(n)$ **linear** — searching an unsorted list
- $O(n \log n)$ **loglinear** — heapsort, quicksort (best and average), merge sort
- $O(n^2)$ **quadratic** — bubble sort (worst case or naive implementation), Shell sort, quicksort (worst case), selection sort, insertion sort
- $O(n^c)$ **polynomial**
- $O(c^n)$ **exponential** — travelling salesman problem via dynamic programming, determining if two logical statements are equivalent by brute force
- $O(n!)$ **factorial** — TSP via brute force.

Program Complexity

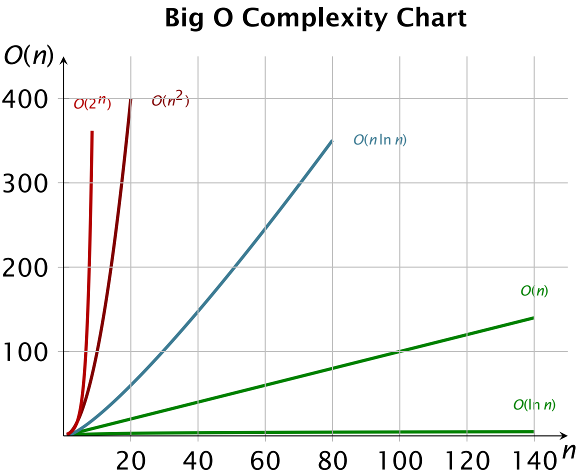
Tyranny of Asymptotics

- Table from Bentley (1984, page 868)
- Cubic algorithm on Cray-1 $3.0n^3$ nanoseconds
- Linear algorithm on TRS-80 $19.5n \times 10^6$ nanoseconds

N	Cray-1	TRS-80
10	3.0 microsecs	200 millisecs
100	3.0 millisecs	2.0 secs
1000	3.0 secs	20 secs
10000	49 mins	3.2 mins
100000	35 days	32 mins
1000000	95 yrs	5.4 hrs

Program Complexity

Big O Complexity Chart



Program Complexity

Big O Notation

- Abuse of notation — we write $f(x) = O(g(x))$
- but $O(g(x))$ is the class of all functions $h(x)$ such that $|h(x)| \leqslant C|g(x)|$ for some constant C
- So we should write $f(x) \in O(g(x))$ (but we don't)
- We ought to use a notation that says that (informally) the function f is bounded both above and below by g asymptotically
- This would mean that for big enough x we have
  $k_1\, g(x) \leqslant f(x) \leqslant k_2\, g(x)$ for some k_1, k_2
- This is Big Theta, $f(x) = \Theta(g(x))$
- But we use Big O to indicate an asymptotically tight bound where Big Theta might be more appropriate
- See Wikipedia: Big O Notation
- This could be Maths phobia generated confusion

Program Complexity

Example

```python
def someFunction(aList) :
  n = len(aList)
  best = 0
  for i in range(n) :
    for j in range(i + 1, n + 1) :
      s = sum(aList[i:j])
      best = max(best, s)
  return best
```

- Example from M269 Unit 2 page 46
- Code in file M269TutorialProgPythonADT.py
- What does the code do ?
- (It was a *famous* problem from the late 1970s/early 1980s)
- Can we construct a more efficient algorithm for the same computational problem ?

Program Complexity

Example (2)

- The code calculates the maximum subsegment of a list
- Described in Bentley (1984), (1988, column 7), (2000, column 7) Also in Gries (1989)
- These are all in a procedural programming style (as in C, Java, Python)
- Problem arose from medical image processing.
- A functional approach using Haskell is in Bird (1998, page 134), (2014, page 127, 133) — a variant on this called the *Not the maximum segment sum* is given in Bird (2010, Page 73) — both of these *derive* a linear time program from the (n^3) initial specification
- See Wikipedia: Maximum subarray problem
- See Rosetta Code: Greatest subsequential sum

Program Complexity

Example (3)

- Here is the same program but modified to allow lists that may only have negative numbers
- The complexity $T(n)$ function will be slightly different
- but the Big O complexity will be the same

```python
def maxSubSeg01(xs) :
  n = len(xs)
  maxSoFar = xs[0]
  for i in range(1,n) :
    for j in range(i + 1, n + 1) :
      s = sum(xs[i:j])
      maxSoFar = max(maxSoFar, s)
  return maxSoFar
```

Program Complexity

Example (4)

- Complexity function $T(n)$ for `maxSubSeg01()`
- Two initial assignments
- The outer loop will be executed $(n - 1)$ times,
- Hence the inner loop is executed

$$(n-1) + (n-2) + \ldots + 2 + 1 = \frac{(n-1)}{2} \times n$$

- Assume `sum()` takes n assignments
- Hence $T(n) = 2 + (n+2) \times \left(\frac{(n-1)}{2} \times n \right)$

$$\begin{aligned}
&= 2 + (n+2) \times \left(\frac{n^2}{2} - \frac{n}{2} \right) \\
&= 2 + \tfrac{1}{2} n^3 - \tfrac{1}{2} n^2 + n^2 - n \\
&= \tfrac{1}{2} n^3 + \tfrac{1}{2} n^2 - n + 2
\end{aligned}$$

- Hence $O(n^3)$

Program Complexity

Example (5)

- **Developing a better algorithm**
- Assume we know the solution (`maxSoFar`) for `xs[0..(i - 1)]`
- We extend the solution to `xs[0..i]` as follows:
- The maximum segment will be either `maxSoFar`
- or the sum of a sublist ending at `i` (`maxToHere`) if it is bigger
- This reasoning is similar to divide and conquer in binary search or Dynamic programming (see Unit 5)
- Keep track of both `maxSoFar` and `maxToHere` — the Eureka step

Program Complexity

Example (6)

- **Developing a better algorithm** `maxSubSeg02()`

```python
def maxSubSeg02(xs) :
  maxToHere = xs[0]
  maxSoFar  = xs[0]
  for x in xs[1:] :
    # Invariant: maxToHere, maxSoFar OK for xs[0..(i-1)]
    maxToHere = max(x, maxToHere + x)
    maxSoFar  = max(maxSoFar, maxToHere)
  return maxSoFar
```

- Complexity function $T(n) = 2 + 2n$
- Hence $O(n)$
- What if we want more information ?
- Return the (or a) segment with max sum and position in list

Program Complexity

Example (7)

```python
def maxSubSeg03(xs) :
  maxSoFar = maxToHere = xs[0]
  startIdx, endIdx, startMaxToHere = 0, 0, 0
  for i, x in enumerate(xs) :
    if maxToHere + x < x :
      maxToHere = x
      startMaxToHere = i
    else :
      maxToHere = maxToHere + x

    if maxSoFar < maxToHere :
      maxSoFar = maxToHere
      startIdx, endIdx = startMaxToHere, i

  return (maxSoFar,xs[startIdx:endIdx+1],startIdx,endIdx)
```

- **Developing a better algorithm** `maxSubSeg03()`
- Complexity function worst case $T(n) = 2 + 3 + (2 + 3)n$
- Hence still $O(n)$
- Note Python assignments, enumerate() and tuple

Program Complexity

Example (8)

- Sample data and output

```python
egList = [-2,1,-3,4,-1,2,1,-5,4]

egList01 = [-1,-1,-1]

egList02 = [1,2,3]

assert maxSubSeg03(egList) == (6, [4, -1, 2, 1], 3, 6)

assert maxSubSeg03(egList01) == (-1, [-1], 0, 0)

assert maxSubSeg03(egList02) == (7, [1, 2, 3], 0, 2)
```

Program Complexity

Python Data Types — Lists

Operation	Notation	Average	Amortized Worst
Get item	`x = xs[i]`	$O(1)$	$O(1)$
Set item	`xs[i] = x`	$O(1)$	$O(1)$
Append	`xs = ys + zs`	$O(1)$	$O(1)$
Copy	`xs = ys[:]`	$O(n)$	$O(n)$
Pop last	`xs.pop()`	$O(1)$	$O(1)$
Pop other	`xs.pop(i)`	$O(k)$	$O(k)$
Insert(i,x)	`xs[i:i] = [x]`	$O(n)$	$O(n)$
Delete item	`del xs[i:i+1]`	$O(n)$	$O(n)$
Get slice	`xs = ys[i:j]`	$O(k)$	$O(k)$
Set slice	`xs[i:j] = ys`	$O(k+n)$	$O(k+n)$
Delete slice	`xs[i:j] = []`	$O(n)$	$O(n)$
Member	`x in xs`	$O(n)$	
Get length	`n = len(xs)`	$O(1)$	$O(1)$
Count(x)	`n = xs.count(x)`	$O(n)$	$O(n)$

- Source https://wiki.python.org/moin/TimeComplexity
- See https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

Program Complexity

User Defined Type — Bags

```python
class Bag:

  def __init__(self):
    self.list = []

  def add(self, item):
    self.list.append(item)

  def remove(self, item):
    self.list.remove(item)

  def contains(self, item):
    return item in self.list

  def count(self, item):
    return self.list.count(item)

  def size(self):
    return len(self.list)

  def __str__(self):
    return str(self.list)
```

Using a Data Type

Information Retrieval Functions

- **Term Frequency**, `tf`, takes a string, `term`, and a Bag, `document`

  returns occurrences of `term` divided by total strings in `document`

- **Inverse Document Frequency**, `idf`, takes a string, `term`, and a list of Bags, `documents`

  returns $\log(\texttt{total}/(1 + \texttt{containing}))$ — `total` is total number of Bags, `containing` is the number of Bags containing `term`

- **tf-idf**, `tf_idf`, takes a string, `term`, and a list of Bags, `documents`

  returns a sequence $[r_0, r_1, \ldots, r_{n-1}]$ such that $r_i = \texttt{tf}(\texttt{term}, d_i) \times \texttt{idf}(\texttt{term}, \texttt{documents})$

Complexity

Big-O and Big-Theta Definitions (a)

- We compare the functions implementing algorithms by looking at the asymptotic behaviour of the functions for large inputs.
- If f and g are functions taking taking natural numbers as input (the problem size) and returning nonnegative results (the effort required in the calculations.)
- f is of *order* g and write $f = \Theta(g)$, if there are positive constants k_1 and k_2 and a natural number n_0 such that

  $k_1\, g(n) \leqslant f(n) \leqslant k_2\, g(n)$ for all $n > n_0$

  This means that some multipliers times $g(n)$ provide upper and lower bounds to $f(n)$
- If we only wanted an upper bound on the values of a function, then you can use Big-O notation.
- We say f is of *order at most* g and write $f = O(g)$, if there is a positive constant k and a natural number n_0 such that

  $f(n) \leqslant kg(n)$ for all $n > n_0$

Complexity

Big-O and Big-Theta Definitions (b)

- Note that the notation is heavily abused:

  Many authors use Big-O notation when they really mean Big-Θ notation

  We really should define the Θ notation to say that $\Theta(g)$ denotes the set of all functions f with the stated property and write $f \in \Theta(g)$ — however the use of $f = \Theta(g)$ is traditional
- The next section gives some rules for manipulating the notation to calculate overall complexities of functions from their component parts — this also abuses the notation for equality

  Based on Bird and Gibbons (2020, page 25) *Algorithm Design with Haskell* and Graham, Knuth and Patashnik (1994, page 450) *Concrete Mathematics: A Foundation for Computer Science*

Complexity

Big-O and Big-Theta Rules

- $n^p = O(n^q)$ where $p \leqslant q$

  This has some surprising consequences — $n = O(n)$ and $n = O(n^2)$ — remember Big-O just gives upper bounds.
- $O(f(n)) + O(g(n)) = O(|f(n)| + |g(n)|)$
- $\Theta(n^p) + \Theta(n^q) = \Theta(n^q)$ where $p \leqslant q$
- $f(n) = \Theta(f(n))$
- $c \cdot \Theta(f(n)) = \Theta(f(n))$ if c is constant
- $\Theta(\Theta(f(n))) = \Theta(f(n))$
- $\Theta(f(n))\Theta(g(n)) = \Theta(f(n)g(n))$
- $\Theta(f(n)g(n)) = f(n)\Theta(g(n))$

Complexity

Big-Theta Rules — Example

```python
def numVowels(txt : str) -> int ;
  """Find the number of vowels in text

  """

  vowelCount = 0
  vowels = "aeiouAEIOU"

  for ch in txt :
    if ch in vowels :
      vowelCount = vowelCount + 1
  return vowelCount
```

- The rules give

  $\Theta(1) + \Theta(1) + \Theta(n) \times \Theta(|\text{vowels}|) \times \Theta(1)$

  where $n = |\text{txt}|$
- Since $|\text{vowels}| = 10$ the overall complexity is $\Theta(n)$

List Comprehensions

Python

- **List Comprehensions** (tutorial), **List Comprehensions** (reference) provide a concise way of performing calculations over lists (or other iterables)
- Example: Square the even numbers between 0 and 9

```
Python3>>> [x ** 2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
```

  - Example: List all pairs of integers (x, y) such that $x < 4$, $y < 4$ and x is divisible by 2 and y is divisible by 3

```
Python3>>> [(x,y) for x in range(4)
...              for y in range(4)
...              if x % 2 == 0
...                 and y % 3 == 0]
[(0, 0), (0, 3), (2, 0), (2, 3)]
Python3>>>
```

- In general

```
  [expr for target1 in iterable1 if cond1
        for target2 in iterable2 if cond2 ...
        for targetN in iterableN if condN ]
```

- Lots example usage in the algorithms below

List Comprehensions

Haskell

- **List Comprehensions** provide a concise way of performing calculations over lists
- Example: Square the even numbers between 0 and 9

```
GHCi> [x^2 | x <- [0..9], x `mod` 2 == 0]
[0,4,16,36,64]
GHCi>
```

- In general

```
[expr | qual1, qual2,..., qualN]
```

- The qualifiers `qual` can be
  - Generators `pattern <- list`
  - Boolean guards — acting as filters
  - Local declarations with `let decls` for use in `expr` and later generators and boolean guards

List Comprehension Exercises

Activity 1 (a) Stop Words Filter

- Stop words are the most common words that most search engines avoid: `'a','an','the','that',...`
- Using list comprehensions, write a function `filterStopWords` that takes a list of words and filters out the stop words
- Here is the initial code

```
sentence \
 = "the␣quick␣brown␣fox␣jumps␣over␣the␣lazy␣dog"

words = sentence.split()

wordsTest \
 = (words == ['the', 'quick', 'brown'
             , 'fox', 'jumps', 'over'
             , 'the', 'lazy', 'dog'])

stopWords \
 = ['a','an','the','that']
```

▸ Go to Answer

List Comprehension Exercises

Activity 1 (a) Stop Words Filter

```
sentence \
 = "the␣quick␣brown␣fox␣jumps␣over␣the␣lazy␣dog"

words = sentence.split()

wordsTest \
 = (words == ['the', 'quick', 'brown'
             , 'fox', 'jumps', 'over'
             , 'the', 'lazy', 'dog'])

stopWords \
 = ['a','an','the','that']
```

- Notice the Python Explicit line joining with (\<nl>) and Python Implicit line joining with ((...))
- The backslash (\) must be followed by an end of line character (<nl>)
- The ('␣') symbol represents a space (see Unicode U+2423 Open Box)

Go to Answer

List Comprehension Exercises

Activity 1 (b) Transpose Matrix

- A matrix can be represented as a list of rows of numbers
- We *transpose* a matrix by swapping columns and rows
- Here is an example

```
matrixA \
 = [[1, 2, 3, 4]
   ,[5, 6, 7 ,8]
   ,[9, 10, 11, 12]]

matATr \
 = [[1, 5, 9]
   ,[2, 6, 10]
   ,[3, 7, 11]
   ,[4, 8, 12]]
```

- Using list comprehensions, write a function `transMat`, to transpose a matrix

Go to Answer

List Comprehension Exercises

Activity 1 (c) List Pairs in Fair Order

- Write a function which takes a pair of positive integers and outputs a list of all possible pairs in those ranges
- If we do this in the simplest way we get a bias to one argument
- Here is an example of a bias to the second argument

```
yBiasLstTest \
 = (yBiasListing(5,5)
    == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
        , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
        , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
        , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
        , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Answer

List Comprehension Exercises

Activity 1 (c) List Pairs in Fair Order

- Rewrite the function which takes a pair of positive integers and outputs a list of all possible pairs in those ranges
- The output should treat each argument *fairly* — any initial prefix should have roughly the same number of instances of each argument
- Here is an example output

```
fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
         , (0, 1), (1, 0)
         , (0, 2), (1, 1), (2, 0)
         , (0, 3), (1, 2), (2, 1), (3, 0)
         , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])
```

Go to Answer

List Comprehension Exercises

Activity 1 (c) List Pairs in Fair Order

- Rewrite the function which takes a pair of positive integers and outputs a list of lists of all possible pairs in those ranges
- The output should treat each argument *fairly* — any initial prefix should have roughly the same number of instances of each argument — further, the output should be segment by each initial prefix (see example below)
- Here is an example output

```
fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
        , [(0, 1), (1, 0)]
        , [(0, 2), (1, 1), (2, 0)]
        , [(0, 3), (1, 2), (2, 1), (3, 0)]
        , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

▸ Go to Answer

Compexity

List Comprehensions

- Note that list comprehensions are not in M269
- See Complexity of a List Comprehension

```
[f(e) for e in row for row in mat]
```

- Suppose $f = \Theta(g)$ with n elements in a row and m rows
- Then complexity is
  $\Theta(g(e)) \times \Theta(n) \times \Theta(m) = \Theta(m \times n \times g(e))$

```
[[e**2 for e in row] for row in mat]
```

- $\Theta(e * *2) = \Theta(1)$
- Suppose n is maximum length of a row and m rows
- Then complexity is$\Theta(1) \times \Theta(n) \times \Theta(m) = \Theta(n \times m)$

List Comprehension Exercises

Answer 1 (a) Stop Words Filter

- Answer 1 (a) Stop Words Filter
- *Write here:*
- Answer 1 continued on next slide

Go to Activity

List Comprehension Exercises

Answer 1 (a) Stop Words Filter

- Answer 1 (a) Stop Words Filter

```
def filterStopWords(words) :
  nonStopWords \
   = [word for word in words
           if word not in stopWords]
  return nonStopWords


filterStopWordsTest \
 = filterStopWords(words) \
    == ['quick', 'brown', 'fox'
      , 'jumps', 'over', 'lazy', 'dog']
```

Go to Activity

List Comprehension Exercises

Answer 1 (b) Transpose Matrix

- Answer 1 (b) Transpose Matrix
- *Write here:*
- Answer 1 continued on next slide

Go to Activity

List Comprehension Exercises

Answer 1 (b) Transpose Matrix

- Answer 1 (b) Transpose Matrix

```python
def transMat(mat) :
  rowLen = len(mat[0])
  matTr \
   = [[row[i] for row in mat] for i in range(rowLen)]
  return matTr

transMatTestA \
 = (transMat(matrixA)
     == matATr)
```

- Note that a list comprehension is a valid expression as a target expression in a list comprehension
- The code assumes every row is of the same length
- Answer 1 continued on next slide

Go to Activity

List Comprehension Exercises

Answer 1 (b) Transpose Matrix

- Note the differences in the list comprehensions below

```
matrixA \
 = [[1, 2, 3, 4]
   ,[5, 6, 7 ,8]
   ,[9, 10, 11, 12]]
```

```
Python3>>> [[row[i] for row in matrixA]
...                 for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
Python3>>> [row[i] for row in matrixA
...                for i in range(4)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Python3>>> [row[i] for i in range(4)
...                for row in matrixA]
[1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8, 12]
Python3>>> [[row[i] for i in range(4)]
...                 for row in matrixA]
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

▸ Go to Activity

List Comprehension Exercises

Answer 1 (b) Transpose Matrix

- Answer 1 (b) Transpose Matrix
- The Python NumPy package provides functions for N-dimensional array objects
- For transpose see numpy.ndarray.transpose

```
Python3>>> import numpy as np
Python3>>> ar = np.array([[1,2],[3,4]])
Python3>>> ar
array([[1, 2],
       [3, 4]])
Python3>>> arT = ar.transpose()
Python3>>> arT
array([[1, 3],
       [2, 4]])
Python3>>> ar
array([[1, 2],
       [3, 4]])
Python3>>> ar.shape
(2, 2)
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — first version
- Write here

```
yBiasLstTest \
 = (yBiasListing(5,5)
     == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
         , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
         , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
         , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
         , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order
- This is the *obvious* but biased version

```python
def yBiasListing(xRng,yRng) :
  yBiasLst \
   = [(x,y) for x in range(xRng)
            for y in range(yRng)]
  return yBiasLst

yBiasLstTest \
 = (yBiasListing(5,5)
     == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
        , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
        , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
        , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
        , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — second version
- Write here

```
fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
         , (0, 1), (1, 0)
         , (0, 2), (1, 1), (2, 0)
         , (0, 3), (1, 2), (2, 1), (3, 0)
         , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — second version
- This works by making the sum of the coordinates the same for each prefix

```
def fairListing(xRng,yRng) :
  fairLst \
   = [(x,d-x) for d in range(yRng)
              for x in range(d+1)]
  return fairLst

fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
        , (0, 1), (1, 0)
        , (0, 2), (1, 1), (2, 0)
        , (0, 3), (1, 2), (2, 1), (3, 0)
        , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — third version
- Write here

```
fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
         , [(0, 1), (1, 0)]
         , [(0, 2), (1, 1), (2, 0)]
         , [(0, 3), (1, 2), (2, 1), (3, 0)]
         , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

Go to Activity

List Comprehension Exercises

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — third version
- The *inner loop* is placed into its own list comprehension

```
def fairListingA(xRng,yRng) :
  fairLstA \
   = [[(x,d-x) for x in range(d+1)]
               for d in range(yRng)]
  return fairLstA

fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
         , [(0, 1), (1, 0)]
         , [(0, 2), (1, 1), (2, 0)]
         , [(0, 3), (1, 2), (2, 1), (3, 0)]
         , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

Go to Activity

- **The Divide-and-Conquer Method**

  Many useful algorithms are recursive in structure and often follow a *divide-and-conquer method*

  They break the problem into several subproblems similar to the original problem
- The time analysis is represented by a *recurrence system*
- **References**
- Big O notation
- Master theorem
- Cormen et al (2022, chp 4) *Algorithms*
- These notes are partly based on *M261 Mathematics in Computing* and *M263 Building Blocks of Software* and are *not* part of *M269 Algorithms, Data Structures and Computability*

Master Theorem

Recurrence System (a)

- **Recurrence System**

$$T(1) = b \quad (1)$$

$$T(n) = bn^{\beta} + cT\left(\frac{n}{d}\right) \qquad \{n = d^{\alpha} > 1\} \quad (2)$$

- **Typical Expansion**

n	**T(n)**
d^0	b
d^1	$bn^{\beta} + cb$
d^2	$bn^{\beta} + cb\left(\frac{n}{d}\right)^{\beta} + c^2 b$

Master Theorem

Recurrence System (b)

- **General Expansion**

$$
\begin{aligned}
T(n) &= bn^{\beta} + cT\left(\frac{n}{d}\right) \\
&= bn^{\beta} + cb\left(\frac{n}{d}\right)^{\beta} + c^2 T\left(\frac{n}{d^2}\right) \\
&= bn^{\beta}\left(1 + \frac{c}{d^{\beta}} + \left(\frac{c}{d^{\beta}}\right)^2 + \cdots + \left(\frac{c}{d^{\beta}}\right)^{\alpha}\right) \\
T(n) &= bn^{\beta} \sum_{i=0}^{\log_d n} \left(\frac{c}{d^{\beta}}\right)^i \qquad (3)
\end{aligned}
$$

Master Theorem

Recurrence System (c)

- **Proof of Closed Form Equation (3)**
- For $n = 1$ equation (3) gives

  $T(1) = b1^{\beta} \sum_{i=0}^{0} \left(\frac{c}{d^{\beta}}\right)^{i} = b$ which is correct (same as (1))
- Assume equation (3) holds for $n = d^{\alpha}$. Then for $n = d^{\alpha+1}$

$$
\begin{aligned}
T\left(d^{\alpha+1}\right) &= cT\left(d^{\alpha}\right) + bn^{\beta} && \text{by equation (2)} \\
&= cbd^{\alpha\beta} \sum_{i=0}^{\alpha} \left(\frac{c}{d^{\beta}}\right)^{i} + bd^{(\alpha+1)\beta} && \text{by assumption} \\
&= \left(\frac{c}{d^{\beta}}\right) bd^{(\alpha+1)\beta} \sum_{i=0}^{\alpha} \left(\frac{c}{d^{\beta}}\right)^{i} + bd^{(\alpha+1)\beta} \\
&= bd^{(\alpha+1)\beta} \left(\sum_{i=1}^{\alpha+1} \left(\frac{c}{d^{\beta}}\right)^{i} + 1 \right) && \text{by rearrangement} \\
&= bd^{(\alpha+1)\beta} \sum_{i=0}^{\alpha+1} \left(\frac{c}{d^{\beta}}\right)^{i} && \text{by rearrangement}
\end{aligned}
$$

- Hence equation (3) holds for all $n = d^{\alpha}$ where $\alpha \in \mathbb{N}$

Master Theorem

Cases

1. If $c < d^\beta$ then the sum converges and T(n) is $\Theta(n^\beta)$
2. If $c = d^\beta$ then each term in the sum is 1 and T(n) is $\Theta\left(n^\beta \log_d n\right)$
3. If $c > d^\beta$ then use $\sum_{i=0}^{p} x^i = \dfrac{x^{p+1} - 1}{x - 1}$

$$
\begin{aligned}
T(n) &= bn^\beta \left[\frac{\left(\frac{c}{d^\beta}\right)^{\log_d n + 1} - 1}{\left(\frac{c}{d^\beta}\right) - 1} \right] \\
&= \Theta\left(n^\beta \left(\tfrac{c}{d^\beta}\right)^{\log_d n} \right) \\
&= \Theta\left(c^{\log_d n} \right) \\
&= \Theta\left(n^{\log_d c} \right) \text{ since } a^{\log_b x} = x^{\log_b a}
\end{aligned}
$$

Master Theorem Example Usage (1)

Binary Search

- **Algorithm**
- Find mid point and check
  if not equal to target, recurse on half the data
- **Timing equations**

  $T(1) \leqslant 1$

  $T(n) = T\left(\frac{n}{2}\right) + 1$
- Hence $c = 1$, $d = 2$, $\beta = 0 \rightarrow$ case (2)

  $$T(n) = \Theta(\log_2 n)$$

Master Theorem Example Usage (2)

Quicksort

- **Algorithm**
- Best case: splitting on median of data
- Recursively sort each half
- **Timing equations**

  $T(1) \leqslant k$

  $T(n) = 2T\left(\frac{n}{2}\right) + kn$
- Hence $c = 2$, $d = 2$, $\beta = 1 \to$ case (2)

  $$T(n) = \Theta(n \log_2 n)$$
- See Averages/Median

Master Theorem Example Usage (3)

Matrix Multiplication — Strassen's Algorithm (a)

- **Matrix Multiplication**
- Let A, B be two square matrices over a ring, $\mathcal{R}$
- Informally, a *ring* is a set with two binary operations which look similar to addition and multiplication of integers
- The problem is to implement matrix multiplication to find the matrix product $C = AB$
- Without loss of generality, we may assume that A, and B have sizes which are powers of 2 — if A, and B were not of this size, they could be padded with rows or columns of zeroes
- The Strassen algorithm partitions A, B and C into equally sized blocks

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

with $A_{ij}, B_{ij}, C_{ij} \in \mathrm{Mat}_{2^{n-1} \times 2^{n-1}}(\mathcal{R})$

Master Theorem Example Usage (3)

Matrix Multiplication — Strassen's Algorithm (b)

- The usual (naive, standard) algorithm gives
$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$
$$= \begin{pmatrix} A_{11} \times B_{11} + A_{12} \times B_{21} & A_{11} \times B_{12} + A_{12} \times B_{22} \\ A_{21} \times B_{11} + A_{22} \times B_{21} & A_{21} \times B_{12} + A_{22} \times B_{22} \end{pmatrix}$$
- This as 8 multiplications and if we assume multiplication is more expensive than addition then the time complexity is $\Theta(n^3)$

Master Theorem Example Usage (3)

Matrix Multiplication — Strassen's Algorithm (c)

- The Strassen algorithm rearranges the calculation

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A22) \times B_{11} \\
M_3 &= A_{11} \times (B_{12} - B_{22}) \\
M_4 &= A_{22} \times (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A12) \times B_{22} \\
M_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22})
\end{aligned}
$$

- We now express the C_{ij} in terms of the M_k

$$
\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}
= \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix}
$$

Master Theorem Example Usage (3)

Matrix Multiplication — Strassen's Algorithm (d)

- **Strassen Matrix Multiplication Timing Equations**

  $T(n) = 7T\left(\frac{n}{2}\right) + \frac{18}{4}n^2$

  $T(1) \leqslant \frac{18}{4}$

- This is derived from the 7 multiplications and 18 additions or subtractions
- $c = 7$, $d = 2$, $\beta = 2 \rightarrow$ case (3)

  $$T(n) = \Theta\left(n^{\log_2 7}\right) = \Theta\left(n^{2.8}\right)$$

Exponentials and Logarithms

Definitions

- **Exponential function** $y = a^x$ or $f(x) = a^x$
- $a^n = a \times a \times \cdots \times a$ (n a terms)
- **Logarithm** reverses the operation of exponentiation
- $\log_a y = x$ means $a^x = y$
- $\log_a 1 = 0$
- $\log_a a = 1$
- Method of logarithms propounded by John Napier from 1614
- Log Tables from 1617 by Henry Briggs
- Slide Rule from about 1620–1630 by William Oughtred of Cambridge
- *Logarithm* from Greek *logos* ratio, and *arithmos* number *Chambers Dictionary (13th Edition, 2014)*

Exponentiation

Rules of Indices

1. $a^m \times a^n = a^{m+n}$
2. $a^m \div a^n = a^{m-n}$
3. $a^{-m} = \dfrac{1}{a^m}$
4. $a^{\frac{1}{m}} = \sqrt[m]{a}$
5. $(a^m)^n = a^{mn}$
6. $a^{\frac{n}{m}} = \sqrt[m]{a^n}$
7. $a^0 = 1$ where $a \neq 0$

- **Exercise** Justify the above rules
- What should 0^0 evaluate to ?
- See Wikipedia: Exponentiation
- The *justification* above probably only worked for whole or *rational* numbers — see later for exponents with *real* numbers (and the value of *logarithms*, *calculus*. . .)

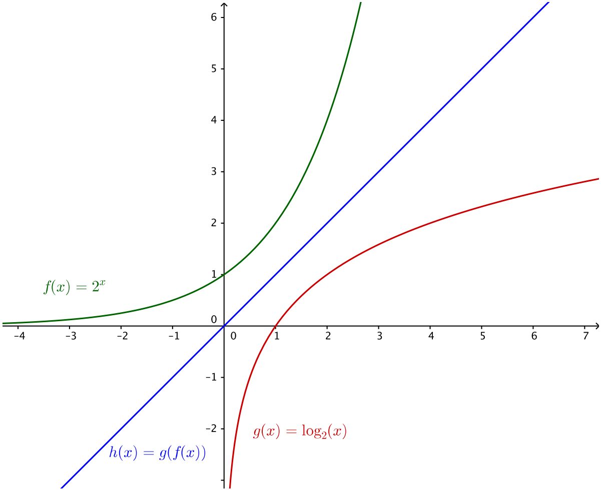
Logarithms

Motivation

- Make arithmetic easier — turns multiplication and division into addition and subtraction (see later)
- Complete the range of elementary functions for differentiation and integration
- An elementary function is a function of one variable which is the composition of a finite number of arithmetic operations ((+), (–), (×), (÷)), exponentials, logarithms, constants, and solutions of algebraic equations (a generalization of nth roots).
- The elementary functions include the trigonometric and hyperbolic functions and their inverses, as they are expressible with complex exponentials and logarithms.
- See A Level FP2 for Euler's relation $\mathrm{e}^{i\theta} = \cos\theta + \mathrm{i}\sin\theta$
- In A Level C3, C4 we get $\int \frac{1}{x} = \log_{\mathrm{e}} |x| + C$
- e is Euler's number 2.71828...

Exponentials and Logarithms
Graphs

- See GeoGebra file expLog.ggb



Exponentials and Logarithms

Laws of Logarithms

- **Multiplication law** $\log_a xy = \log_a x + \log_a y$
- **Division law** $\log_a \left(\frac{x}{y}\right) = \log_a x - \log_a y$
- **Power law** $\log_a x^k = k \log_a x$
- **Proof of Multiplication Law**

$$
\begin{aligned}
x &= a^{\log_a x} \\
y &= a^{\log_a y} && \text{by definition of log} \\
xy &= a^{\log_a x} \times a^{\log_a y} \\
&= a^{\log_a x + \log_a y} && \text{by laws of indices} \\
\text{Hence } \log_a xy &= \log_a x + \log_a y && \text{by definition of log}
\end{aligned}
$$

Arithmetic Operations

Inverse Operations

- *Notation helps or maybe not ?*
- **Addition** $\text{add}(b, x) = x + b$
- **Subtraction** $\text{sub}(b, x) = x - b$
- Inverse $\text{sub}(b, \text{add}(b, x)) = (x + b) - b = x$
- **Multiplication** $\text{mul}(b, x) = x \times b$
- **Division** $\text{div}(b, x) = x \div b = \frac{x}{b} = x/b$
- Inverse $\text{div}(b, \text{mul}(b, x)) = (x \times b) \div b = \frac{(x \times b)}{b} = x$
- **Exponentiation** $\exp(b, x) = b^x$
- **Logarithm** $\log(b, x) = \log_b x$
- Inverse $\log(b, \exp(b, x)) = \log_b(b^x) = x$
- *What properties do the operations have that work (or not) with the notation ?*

Arithmetic Operations

Commutativity and Associativity

- **Commutativity** $x \circledast y = y \circledast x$
- **Associativity** $(x \circledast y) \circledast z = x \circledast (y \circledast z)$
- $(+)$ and $(\times)$ are *semantically* commutative and associative — so we can leave the brackets out
- $(-)$ and $(\div)$ are not
- Evaluate $(3 - (2 - 1))$ and $((3 - 2) - 1)$
- Evaluate $(3/(2/2))$ and $((3/2)/2)$
- We have the *syntactic* ideas of left (and right) associativity
- We choose $(-)$ and $(\div)$ to be left associative
- $3 - 2 - 1$ means $((3 - 2) - 1)$
- $3/2/2$ means $((3/2)/2)$
- **Operator precedence** is also a choice (remember BIDMAS or BODMAS ?)
- If in doubt, put the brackets in

Exponentials and Logarithms

Associativity

- What should 2^{3^4} mean ?
- Let $b \wedge x \equiv b^x$
- Evaluate $(2 \wedge 3) \wedge 4$ and $2 \wedge (3 \wedge 4)$
- Evaluate $c = \log_b(\log_b((b \wedge b) \wedge x))$
- Evaluate $d = \log_b(\log_b(b \wedge (b \wedge x)))$
- Beware spreadsheets Excel and LibreOffice here

Exponentials and Logarithms

Associativity

- $(2^3)^4 = 2^{12}$ and $2^{3^4} = 2^{81}$
- Exponentiation is not semantically associative
- We *choose* the syntactic left or right associativity to make the syntax nicer.
- Evaluate $c = \log_b(\log_b((b \wedge b) \wedge x))$
- $c = \log_b(x \log_b(b^b)) = \log_b(x \cdot (b \log_b b)) = \log_b(x \cdot b \cdot 1)$
- Hence $c = \log_b x + \log_b b = \log_b x + 1$
- Not symmetrical (unless b and x are both 2)
- Evaluate $d = \log_b(\log_b(b \wedge (b \wedge x)))$
- $d = \log_b((b \wedge x)(\log_b b)) = \log_b((b \wedge x) \times 1)$
- Hence $d = \log_b(b \wedge x) = x(\log_b b) = x$
- Which is what we want — so exponentiation is *chosen* to be right associative

Exponentials and Logarithms

Change of Base

- **Change of base**

$$\log_a x = \frac{\log_b x}{\log_b a}$$

Proof: Let $y = \log_a x$

$$a^y = x$$

$$\log_b a^y = \log_b x$$

$$y \log_b a = \log_b x$$

$$y = \frac{\log_b x}{\log_b a}$$

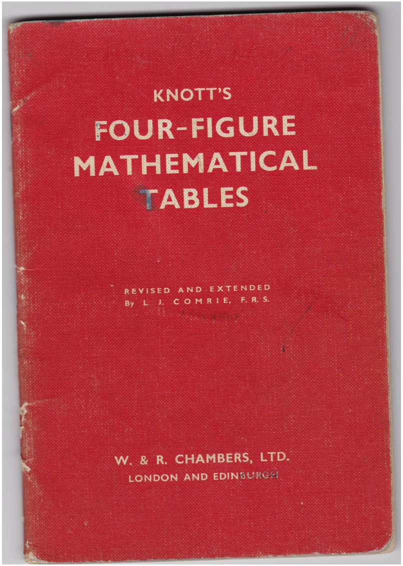
- Given x, $\log_b x$, find the base b
  - $b = x^{\frac{1}{\log_b x}}$
- $\log_a b = \dfrac{1}{\log_b a}$

Before Calculators and Computers

- We had computers before 1950 — they were *humans* with pencil, paper and some further aids:
- **Slide rule** invented by William Oughtred in the 1620s — major calculating tool until pocket calculators in 1970s
- **Log tables** in use from early 1600s — method of logarithms propounded by John Napier
- **Logarithm** from Greek *logos* ratio, and *arithmos* number

Log Tables

Knott's Four-Figure Mathematical Tables



Log Tables

Logarithms of Numbers

x	0	1	2	3	4	5	6	7	8	9	Δ	ADD 1	2	3	4	5	6	7	8	9
10	·0000	0043	0086	0128	0170	0212					42	4	8	13	17	21	25	29	34	38
						0212	0253	0294	0334	0374	40	4	8	12	16	20	24	28	32	36
11	·0414	0453	0492	0531	0569	0607					39	4	8	12	16	19	23	27	31	35
						0607	0645	0682	0719	0755	37	4	7	11	15	18	22	26	30	33
12	·0792	0828	0864	0899	0934	0969					35	4	7	11	14	18	21	25	28	32
						0969	1004	1038	1072	1106	34	3	7	10	14	17	20	24	27	31
13	·1139	1173	1206	1239	1271	1303					33	3	7	10	13	16	20	23	26	30
						1303	1335	1367	1399	1430	32	3	6	10	13	16	19	22	26	29
14	·1461	1492	1523	1553	1584	1614	1644	1673	1703	1732	30	3	6	9	12	15	18	21	24	27
15	·1761	1790	1818	1847	1875	1903	1931	1959	1987	2014	28	3	6	8	11	14	17	20	22	25
16	·2041	2068	2095	2122	2148	2175	2201	2227	2253	2279	26	3	5	8	10	13	16	18	21	23
17	·2304	2330	2355	2380	2405	2430	2455	2480	2504	2529	25	2	5	7	10	12	15	17	20	22
18	·2553	2577	2601	2625	2648	2672	2695	2718	2742	2765	23	2	5	7	9	12	14	16	19	21
19	·2788	2810	2833	2856	2878	2900	2923	2945	2967	2989	22	2	4	7	9	11	13	15	18	20
20	·3010	3032	3054	3075	3096	3118	3139	3160	3181	3201	21	2	4	6	8	11	13	15	17	19
21	·3222	3243	3263	3284	3304	3324	3345	3365	3385	3404	20	2	4	6	8	10	12	14	16	18
22	·3424	3444	3464	3483	3502	3522	3541	3560	3579	3598	19	2	4	6	8	10	11	13	15	17
23	·3617	3636	3655	3674	3692	3711	3729	3747	3766	3784		2	4	6	7	9	11	13	15	17
24	·3802	3820	3838	3856	3874	3892	3909	3927	3945	3962	18	2	4	5	7	9	11	13	14	16
25	·3979	3997	4014	4031	4048	4065	4082	4099	4116	4133	17	2	3	5	7	9	10	12	14	15
26	·4150	4166	4183	4200	4216	4232	4249	4265	4281	4298		2	3	5	6	8	10	11	13	14
27	·4314	4330	4346	4362	4378	4393	4409	4425	4440	4456	16	2	3	5	6	8	9	11	13	14
28	·4472	4487	4502	4518	4533	4548	4564	4579	4594	4609	15	2	3	5	6	8	9	11	12	14
29	·4624	4639	4654	4669	4683	4698	4713	4728	4742	4757		1	3	4	6	7	9	10	12	13
30	·4771	4786	4800	4814	4829	4843	4857	4871	4886	4900	14	1	3	4	6	7	8	10	11	13
31	·4914	4928	4942	4955	4969	4983	4997	5011	5024	5038		1	3	4	5	7	8	10	11	12
32	·5051	5065	5079	5092	5105	5119	5132	5145	5159	5172		1	3	4	5	7	8	9	11	12
33	·5185	5198	5211	5224	5237	5250	5263	5276	5289	5302	13	1	3	4	5	7	8	9	10	12
34	·5315	5328	5340	5353	5366	5378	5391	5403	5416	5428		1	3	4	5	6	8	9	10	11
35	·5441	5453	5465	5478	5490	5502	5514	5527	5539	5551		1	2	4	5	6	7	8	10	11
36	·5563	5575	5587	5599	5611	5623	5635	5647	5658	5670	12	1	2	4	5	6	7	8	10	11
37	·5682	5694	5705	5717	5729	5740	5752	5763	5775	5786		1	2	3	5	6	7	8	9	10
38	·5798	5809	5821	5832	5843	5855	5866	5877	5888	5899		1	2	3	5	6	7	8	9	10
39	·5911	5922	5933	5944	5955	5966	5977	5988	5999	6010	11	1	2	3	4	6	7	8	9	10
40	·6021	6031	6042	6053	6064	6075	6085	6096	6107	6117		1	2	3	4	5	6	7	9	10
41	·6128	6138	6149	6160	6170	6180	6191	6201	6212	6222		1	2	3	4	5	6	7	8	9
42	·6232	6243	6253	6263	6274	6284	6294	6304	6314	6325		1	2	3	4	5	6	7	8	9
43	·6335	6345	6355	6365	6375	6385	6395	6405	6415	6425	10	1	2	3	4	5	6	7	8	9
44	·6435	6444	6454	6464	6474	6484	6493	6503	6513	6522		1	2	3	4	5	6	7	8	9
45	·6532	6542	6551	6561	6571	6580	6590	6599	6609	6618		1	2	3	4	5	6	7	8	9
46	·6628	6637	6646	6656	6665	6675	6684	6693	6702	6712		1	2	3	4	5	6	7	7	8
47	·6721	6730	6739	6749	6758	6767	6776	6785	6794	6803		1	2	3	4	5	5	6	7	8
48	·6812	6821	6830	6839	6848	6857	6866	6875	6884	6893	9	1	2	3	4	5	5	6	7	8
49	·6902	6911	6920	6928	6937	6946	6955	6964	6972	6981		1	2	3	4	4	5	6	7	8

USEFUL CONSTANTS WITH THEIR LOGARITHMS

	No.	Log.		No.	Log.		No.	Log.
π	3·14159	0·4971	1 radian	57°·296	1·7581	e	2·71828	0·4343
$\frac{1}{\pi}$	0·3183	$\bar{1}$·5029	1 radian	3437′·7	3·5363	M	0·4343	$\bar{1}$·6378
			1 radian	206265″	5·3144	$\frac{1}{M}$	2·3026	0·3622
π^2	9·8696	0·9943	arc 1°	0·017 453 293	$\bar{2}$·2419			
$\sqrt{\pi}$	1·7725	0·2486	arc 1′	0·000 290 888	$\bar{4}$·4637			
$\frac{4}{3}\pi$	4·1888	0·6221	arc 1″	0·000 004 848	$\bar{6}$·6856			

$\log_e x = \frac{1}{M} . \log_{10} x$

$\log_{10} x = M . \log_e x$

x	0	1	2	3	4	5	6	7	8	9	Δ	ADD 1	2	3	4	5	6	7	8	9
50	·6990	6998	7007	7016	7024	7033	7042	7050	7059	7067		1	2	3	3	4	5	6	7	8
51	·7076	7084	7093	7101	7110	7118	7126	7135	7143	7152		1	2	3	3	4	5	6	7	8
52	·7160	7168	7177	7185	7193	7202	7210	7218	7226	7235		1	2	2	3	4	5	6	7	7
53	·7243	7251	7259	7267	7275	7284	7292	7300	7308	7316		1	2	2	3	4	5	6	6	7
54	·7324	7332	7340	7348	7356	7364	7372	7380	7388	7396	8	1	2	2	3	4	5	6	6	7
55	·7404	7412	7419	7427	7435	7443	7451	7459	7466	7474		1	2	2	3	4	5	5	6	7
56	·7482	7490	7497	7505	7513	7520	7528	7536	7543	7551		1	2	2	3	4	5	5	6	7
57	·7559	7566	7574	7582	7589	7597	7604	7612	7619	7627		1	2	2	3	4	5	5	6	7
58	·7634	7642	7649	7657	7664	7672	7679	7686	7694	7701		1	2	2	3	4	4	5	6	7
59	·7709	7716	7723	7731	7738	7745	7752	7760	7767	7774		1	1	2	3	4	4	5	6	7
60	·7782	7789	7796	7803	7810	7818	7825	7832	7839	7846	7	1	1	2	3	4	4	5	6	6
61	·7853	7860	7868	7875	7882	7889	7896	7903	7910	7917		1	1	2	3	4	4	5	6	6
62	·7924	7931	7938	7945	7952	7959	7966	7973	7980	7987		1	1	2	3	3	4	5	6	6
63	·7993	8000	8007	8014	8021	8028	8035	8041	8048	8055		1	1	2	3	3	4	5	6	6
64	·8062	8069	8075	8082	8089	8096	8102	8109	8116	8122		1	1	2	3	3	4	5	5	6
65	·8129	8136	8142	8149	8156	8162	8169	8176	8182	8189		1	1	2	3	3	4	5	5	6
66	·8195	8202	8209	8215	8222	8228	8235	8241	8248	8254		1	1	2	3	3	4	5	5	6
67	·8261	8267	8274	8280	8287	8293	8299	8306	8312	8319		1	1	2	3	3	4	4	5	6
68	·8325	8331	8338	8344	8351	8357	8363	8370	8376	8382		1	1	2	3	3	4	4	5	6
69	·8388	8395	8401	8407	8414	8420	8426	8432	8439	8445		1	1	2	3	3	4	4	5	6
70	·8451	8457	8463	8470	8476	8482	8488	8494	8500	8506		1	1	2	2	3	4	4	5	6
71	·8513	8519	8525	8531	8537	8543	8549	8555	8561	8567	6	1	1	2	2	3	4	4	5	5
72	·8573	8579	8585	8591	8597	8603	8609	8615	8621	8627		1	1	2	2	3	4	4	5	5
73	·8633	8639	8645	8651	8657	8663	8669	8675	8681	8686		1	1	2	2	3	4	4	5	5
74	·8692	8698	8704	8710	8716	8722	8727	8733	8739	8745		1	1	2	2	3	4	4	5	5
75	·8751	8756	8762	8768	8774	8779	8785	8791	8797	8802		1	1	2	2	3	3	4	5	5
76	·8808	8814	8820	8825	8831	8837	8842	8848	8854	8859		1	1	2	2	3	3	4	5	5
77	·8865	8871	8876	8882	8887	8893	8899	8904	8910	8915		1	1	2	2	3	3	4	4	5
78	·8921	8927	8932	8938	8943	8949	8954	8960	8965	8971		1	1	2	2	3	3	4	4	5
79	·8976	8982	8987	8993	8998	9004	9009	9015	9020	9025		1	1	2	2	3	3	4	4	5
80	·9031	9036	9042	9047	9053	9058	9063	9069	9074	9079		1	1	2	2	3	3	4	4	5
81	·9085	9090	9096	9101	9106	9112	9117	9122	9128	9133		1	1	2	2	3	3	4	4	5
82	·9138	9143	9149	9154	9159	9165	9170	9175	9180	9186		1	1	2	2	3	3	4	4	5
83	·9191	9196	9201	9206	9212	9217	9222	9227	9232	9238		1	1	2	2	3	3	4	4	5
84	·9243	9248	9253	9258	9263	9269	9274	9279	9284	9289		1	1	2	2	3	3	4	4	5
85	·9294	9299	9304	9309	9315	9320	9325	9330	9335	9340		1	1	2	2	3	3	4	4	5
86	·9345	9350	9355	9360	9365	9370	9375	9380	9385	9390	5	1	1	2	2	3	3	4	4	5
87	·9395	9400	9405	9410	9415	9420	9425	9430	9435	9440		0	1	1	2	2	3	3	4	4
88	·9445	9450	9455	9460	9465	9469	9474	9479	9484	9489		0	1	1	2	2	3	3	4	4
89	·9494	9499	9504	9509	9513	9518	9523	9528	9533	9538		0	1	1	2	2	3	3	4	4
90	·9542	9547	9552	9557	9562	9566	9571	9576	9581	9586		0	1	1	2	2	3	3	4	4
91	·9590	9595	9600	9605	9609	9614	9619	9624	9628	9633		0	1	1	2	2	3	3	4	4
92	·9638	9643	9647	9652	9657	9661	9666	9671	9675	9680		0	1	1	2	2	3	3	4	4
93	·9685	9689	9694	9699	9703	9708	9713	9717	9722	9727		0	1	1	2	2	3	3	4	4
94	·9731	9736	9741	9745	9750	9754	9759	9763	9768	9773		0	1	1	2	2	3	3	4	4
95	·9777	9782	9786	9791	9795	9800	9805	9809	9814	9818		0	1	1	2	2	3	3	4	4
96	·9823	9827	9832	9836	9841	9845	9850	9854	9859	9863		0	1	1	2	2	3	3	4	4
97	·9868	9872	9877	9881	9886	9890	9894	9899	9903	9908		0	1	1	2	2	3	3	4	4
98	·9912	9917	9921	9926	9930	9934	9939	9943	9948	9952		0	1	1	2	2	3	3	4	4
99	·9956	9961	9965	9969	9974	9978	9983	9987	9991	9996	4	0	1	1	2	2	2	3	3	4

Only the decimal portion (*mantissa*) of each logarithm is shown in this table.
The integral portion (*characteristic*) must be determined independently.

Log Tables

Antilogarithms

4 ANTILOGARITHMS

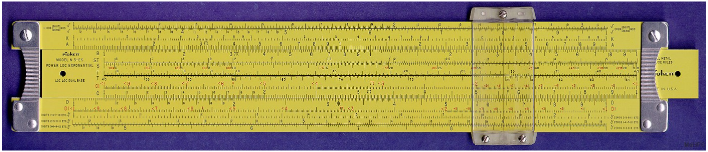
x	0	1	2	3	4	5	6	7	8	9	Δ	ADD 1	2	3	4	5	6	7	8	9
·00	1000	1002	1005	1007	1009	1012	1014	1016	1019	1021	2	0	0	1	1	1	1	1	2	2
·01	1023	1026	1028	1030	1033	1035	1038	1040	1042	1045		0	0	1	1	1	1	2	2	2
·02	1047	1050	1052	1054	1057	1059	1062	1064	1067	1069		0	0	1	1	1	1	2	2	2
·03	1072	1074	1076	1079	1081	1084	1086	1089	1091	1094		0	0	1	1	1	1	2	2	2
·04	1096	1099	1102	1104	1107	1109	1112	1114	1117	1119		0	1	1	1	1	2	2	2	2
·05	1122	1125	1127	1130	1132	1135	1138	1140	1143	1146		0	1	1	1	1	2	2	2	2
·06	1148	1151	1153	1156	1159	1161	1164	1167	1169	1172		0	1	1	1	1	2	2	2	2
·07	1175	1178	1180	1183	1186	1189	1191	1194	1197	1199		0	1	1	1	1	2	2	2	2
·08	1202	1205	1208	1211	1213	1216	1219	1222	1225	1227		0	1	1	1	1	2	2	2	3
·09	1230	1233	1236	1239	1242	1245	1247	1250	1253	1256		0	1	1	1	1	2	2	2	3
·10	1259	1262	1265	1268	1271	1274	1276	1279	1282	1285		0	1	1	1	1	2	2	2	3
·11	1288	1291	1294	1297	1300	1303	1306	1309	1312	1315	3	0	1	1	1	2	2	2	2	3
·12	1318	1321	1324	1327	1330	1334	1337	1340	1343	1346		0	1	1	1	2	2	2	2	3
·13	1349	1352	1355	1358	1361	1365	1368	1371	1374	1377		0	1	1	1	2	2	2	3	3
·14	1380	1384	1387	1390	1393	1396	1400	1403	1406	1409		0	1	1	1	2	2	2	3	3
·15	1413	1416	1419	1422	1426	1429	1432	1435	1439	1442		0	1	1	1	2	2	2	3	3
·16	1445	1449	1452	1455	1459	1462	1466	1469	1472	1476		0	1	1	1	2	2	2	3	3
·17	1479	1483	1486	1489	1493	1496	1500	1503	1507	1510		0	1	1	1	2	2	2	3	3
·18	1514	1517	1521	1524	1528	1531	1535	1538	1542	1545		0	1	1	1	2	2	2	3	3
·19	1549	1552	1556	1560	1563	1567	1570	1574	1578	1581		0	1	1	1	2	2	3	3	3
·20	1585	1589	1592	1596	1600	1603	1607	1611	1614	1618		0	1	1	1	2	2	3	3	3
·21	1622	1626	1629	1633	1637	1641	1644	1648	1652	1656		0	1	1	2	2	2	3	3	3
·22	1660	1663	1667	1671	1675	1679	1683	1687	1690	1694		0	1	1	2	2	2	3	3	3
·23	1698	1702	1706	1710	1714	1718	1722	1726	1730	1734	4	0	1	1	2	2	2	3	3	4
·24	1738	1742	1746	1750	1754	1758	1762	1766	1770	1774		0	1	1	2	2	2	3	3	4
·25	1778	1782	1786	1791	1795	1799	1803	1807	1811	1816		0	1	1	2	2	2	3	3	4
·26	1820	1824	1828	1832	1837	1841	1845	1849	1854	1858		0	1	1	2	2	3	3	3	4
·27	1862	1866	1871	1875	1879	1884	1888	1892	1897	1901		0	1	1	2	2	3	3	3	4
·28	1905	1910	1914	1919	1923	1928	1932	1936	1941	1945		0	1	1	2	2	3	3	4	4
·29	1950	1954	1959	1963	1968	1972	1977	1982	1986	1991		0	1	1	2	2	3	3	4	4
·30	1995	2000	2004	2009	2014	2018	2023	2028	2032	2037		0	1	1	2	2	3	3	4	4
·31	2042	2046	2051	2056	2061	2065	2070	2075	2080	2084		0	1	1	2	2	3	3	4	4
·32	2089	2094	2099	2104	2109	2113	2118	2123	2128	2133		0	1	1	2	2	3	3	4	4
·33	2138	2143	2148	2153	2158	2163	2168	2173	2178	2183	5	1	1	2	2	3	3	4	4	5
·34	2188	2193	2198	2203	2208	2213	2218	2223	2228	2234		1	1	2	2	3	3	4	4	5
·35	2239	2244	2249	2254	2259	2265	2270	2275	2280	2286		1	1	2	2	3	3	4	4	5
·36	2291	2296	2301	2307	2312	2317	2323	2328	2333	2339		1	1	2	2	3	3	4	4	5
·37	2344	2350	2355	2360	2366	2371	2377	2382	2388	2393		1	1	2	2	3	3	4	4	5
·38	2399	2404	2410	2415	2421	2427	2432	2438	2443	2449		1	1	2	2	3	3	4	4	5
·39	2455	2460	2466	2472	2477	2483	2489	2495	2500	2506		1	1	2	2	3	3	4	5	5
·40	2512	2518	2523	2529	2535	2541	2547	2553	2559	2564		1	1	2	2	3	4	4	5	5
·41	2570	2576	2582	2588	2594	2600	2606	2612	2618	2624	6	1	1	2	2	3	4	4	5	5
·42	2630	2636	2642	2649	2655	2661	2667	2673	2679	2685		1	1	2	2	3	4	4	5	6
·43	2692	2698	2704	2710	2716	2723	2729	2735	2742	2748		1	1	2	3	3	4	4	5	6
·44	2754	2761	2767	2773	2780	2786	2793	2799	2805	2812		1	1	2	3	3	4	4	5	6
·45	2818	2825	2831	2838	2844	2851	2858	2864	2871	2877		1	1	2	3	3	4	5	5	6
·46	2884	2891	2897	2904	2911	2917	2924	2931	2938	2944		1	1	2	3	3	4	5	5	6
·47	2951	2958	2965	2972	2979	2985	2992	2999	3006	3013		1	1	2	3	3	4	5	5	6
·48	3020	3027	3034	3041	3048	3055	3062	3069	3076	3083	7	1	1	2	3	4	4	5	6	6
·49	3090	3097	3105	3112	3119	3126	3133	3141	3148	3155		1	1	2	3	4	4	5	6	6

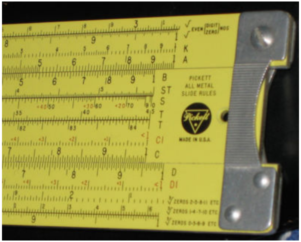
ANTILOGARITHMS 5

x	0	1	2	3	4	5	6	7	8	9	Δ	ADD 1	2	3	4	5	6	7	8	9
·50	3162	3170	3177	3184	3192	3199	3206	3214	3221	3228		1	1	2	3	4	4	5	6	7
·51	3236	3243	3251	3258	3266	3273	3281	3289	3296	3304		1	2	2	3	4	5	5	6	7
·52	3311	3319	3327	3334	3342	3350	3357	3365	3373	3381		1	2	2	3	4	5	5	6	7
·53	3388	3396	3404	3412	3420	3428	3436	3443	3451	3459		1	2	2	3	4	5	6	6	7
·54	3467	3475	3483	3491	3499	3508	3516	3524	3532	3540	8	1	2	2	3	4	5	6	6	7
·55	3548	3556	3565	3573	3581	3589	3597	3606	3614	3622		1	2	2	3	4	5	6	7	7
·56	3631	3639	3648	3656	3664	3673	3681	3690	3698	3707		1	2	3	3	4	5	6	7	8
·57	3715	3724	3733	3741	3750	3758	3767	3776	3784	3793		1	2	3	3	4	5	6	7	8
·58	3802	3811	3819	3828	3837	3846	3855	3864	3873	3882		1	2	3	4	4	5	6	7	8
·59	3890	3899	3908	3917	3926	3936	3945	3954	3963	3972	9	1	2	3	4	5	5	6	7	8
·60	3981	3990	3999	4009	4018	4027	4036	4046	4055	4064		1	2	3	4	5	6	6	7	8
·61	4074	4083	4093	4102	4111	4121	4130	4140	4150	4159		1	2	3	4	5	6	7	8	9
·62	4169	4178	4188	4198	4207	4217	4227	4236	4246	4256		1	2	3	4	5	6	7	8	9
·63	4266	4276	4285	4295	4305	4315	4325	4335	4345	4355	10	1	2	3	4	5	6	7	8	9
·64	4365	4375	4385	4395	4406	4416	4426	4436	4446	4457		1	2	3	4	5	6	7	8	9
·65	4467	4477	4487	4498	4508	4519	4529	4539	4550	4560		1	2	3	4	5	6	7	8	9
·66	4571	4581	4592	4603	4613	4624	4634	4645	4656	4667		1	2	3	4	5	6	7	9	10
·67	4677	4688	4699	4710	4721	4732	4742	4753	4764	4775	11	1	2	3	4	6	7	8	9	10
·68	4786	4797	4808	4819	4831	4842	4853	4864	4875	4887		1	2	3	4	6	7	8	9	10
·69	4898	4909	4920	4932	4943	4955	4966	4977	4989	5000		1	2	3	5	6	7	8	9	10
·70	5012	5023	5035	5047	5058	5070	5082	5093	5105	5117		1	2	4	5	6	7	8	9	11
·71	5129	5140	5152	5164	5176	5188	5200	5212	5224	5236	12	1	2	4	5	6	7	8	10	11
·72	5248	5260	5272	5284	5297	5309	5321	5333	5346	5358		1	2	4	5	6	7	9	10	11
·73	5370	5383	5395	5408	5420	5433	5445	5458	5470	5483		1	3	4	5	6	8	9	10	11
·74	5495	5508	5521	5534	5546	5559	5572	5585	5598	5610		1	3	4	5	6	8	9	10	12
·75	5623	5636	5649	5662	5675	5689	5702	5715	5728	5741	13	1	3	4	5	7	8	9	10	12
·76	5754	5768	5781	5794	5808	5821	5834	5848	5861	5875		1	3	4	5	7	8	9	11	12
·77	5888	5902	5916	5929	5943	5957	5970	5984	5998	6012		1	3	4	5	7	8	10	11	12
·78	6026	6039	6053	6067	6081	6095	6109	6124	6138	6152	14	1	3	4	6	7	8	10	11	13
·79	6166	6180	6194	6209	6223	6237	6252	6266	6281	6295		1	3	4	6	7	9	10	11	13
·80	6310	6324	6339	6353	6368	6383	6397	6412	6427	6442		1	3	4	6	7	9	10	12	13
·81	6457	6471	6486	6501	6516	6531	6546	6561	6577	6592	15	2	3	5	6	8	9	11	12	14
·82	6607	6622	6637	6653	6668	6683	6699	6714	6730	6745		2	3	5	6	8	9	11	12	14
·83	6761	6776	6792	6808	6823	6839	6855	6871	6887	6902		2	3	5	6	8	9	11	13	14
·84	6918	6934	6950	6966	6982	6998	7015	7031	7047	7063	16	2	3	5	6	8	10	11	13	14
·85	7079	7096	7112	7129	7145	7161	7178	7194	7211	7228		2	3	5	7	8	10	12	13	15
·86	7244	7261	7278	7295	7311	7328	7345	7362	7379	7396	17	2	3	5	7	9	10	12	14	15
·87	7413	7430	7447	7464	7482	7499	7516	7534	7551	7568		2	3	5	7	9	10	12	14	16
·88	7586	7603	7621	7638	7656	7674	7691	7709	7727	7745		2	4	5	7	9	11	12	14	16
·89	7762	7780	7798	7816	7834	7852	7870	7889	7907	7925	18	2	4	5	7	9	11	13	14	16
·90	7943	7962	7980	7998	8017	8035	8054	8072	8091	8110		2	4	6	7	9	11	13	15	17
·91	8128	8147	8166	8185	8204	8222	8241	8260	8279	8299	19	2	4	6	8	10	11	13	15	17
·92	8318	8337	8356	8375	8395	8414	8433	8453	8472	8492		2	4	6	8	10	12	14	15	17
·93	8511	8531	8551	8570	8590	8610	8630	8650	8670	8690		2	4	6	8	10	12	14	16	18
·94	8710	8730	8750	8770	8790	8810	8831	8851	8872	8892	20	2	4	6	8	10	12	14	16	18
·95	8913	8933	8954	8974	8995	9016	9036	9057	9078	9099		2	4	6	8	10	12	15	17	19
·96	9120	9141	9162	9183	9204	9226	9247	9268	9290	9311	21	2	4	6	8	11	13	15	17	19
·97	9333	9354	9376	9397	9419	9441	9462	9484	9506	9528		2	4	7	9	11	13	15	17	20
·98	9550	9572	9594	9616	9638	9661	9683	9705	9727	9750	22	2	4	7	9	11	13	15	18	20
·99	9772	9795	9817	9840	9863	9886	9908	9931	9954	9977	23	2	5	7	9	11	14	16	18	21

Slide Rules

Pickett N 3-ES from 1967

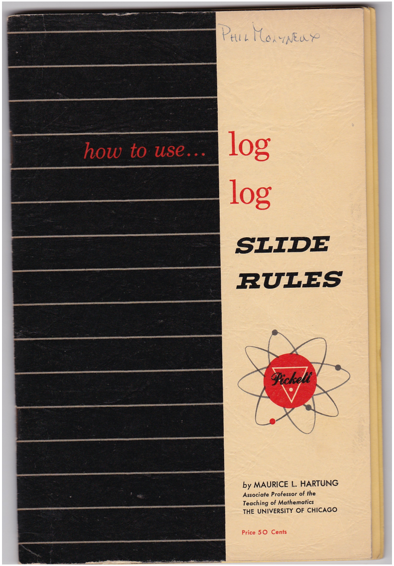




- See Oughtred Society
- UKSRC
- Rod Lovett's Slide Rules
- Slide Rule Museum

Slide Rules

Pickett log log Slide Rules Manual 1953

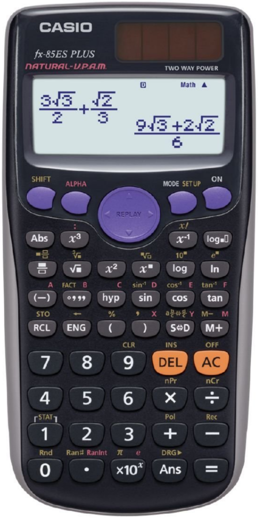


Calculators

HP HP-21 Calculator from 1975 £69

Calculators

Casio fx-85GT PLUS Calculator from 2013 £10



Calculators

Calculator Links

- **HP Calculator Museum** http://www.hpmuseum.org
- **HP Calculator Emulators**
  http://nonpareil.brouhaha.com
- **HP Calculator Emulators for OS X**
  http://www.bartosiak.org/nonpareil/
- **Vintage Calculators Web Museum**
  http://www.vintagecalculators.com

Example Calculation

Log Tables, Slide Rule and Calculator

- Evaluate 89.7×597
- **Knott's Tables**
- $\log_{10} 89.7 = 1.9528$ and $\log_{10} 597 = 2.7760$
- Shows *mantissa* (decimal) & *characteristic* (integral)
- Add 4.7288, take *antilog* to get $5346 + 10 = 5.356 \times 10^4$
- **HP-21 Calculator** — set display to 4 decimal places
- 89.7 $\boxed{\log}$ = 1.9528 and 597 $\boxed{\log}$ = 2.7760
- $\boxed{+}$ displays 4.7288
- 10 $\boxed{\text{ENTER}}$, $\boxed{x \rightleftarrows y}$ and $\boxed{y^x}$ displays 53550.9000
- **Casio fx-85GT PLUS**
- $\boxed{\log}$ 89.7 $\boxed{)}$ = 1.952792443 $\boxed{+}$ $\boxed{\log}$ 597 $\boxed{)}$ = 2.775974331 $\boxed{=}$
- 4.728766774 $\boxed{\text{Ans}}$ + $\boxed{10^x}$ gives 53550.9

Boolean Expressions

Traffic Lights Example (1)

- Consider traffic light at the intersection of roads *AC* and *BD* with the following rules for the *AC* controller
- Vehicles should not wait on red on *BD* for too long.
- If there is a long queue on *AC* then *BD* is only given a green for a short interval.
- If both queues are long the usual flow times are used.
- We use the following propositions:
  - w Vehicles have been waiting on red on *BD* for too long
  - q Queue on *AC* is too long
  - r Queue on *BD* is too long
- Given the following events:
  - `ToBD` Change flow to *BD*
  - `ToBDShort` Change flow to *BD* for short time
  - `NoChange` No Change to lights
- Express above as truth table, outcome tree, boolean expression

Boolean Expressions

Traffic Lights Example (2)

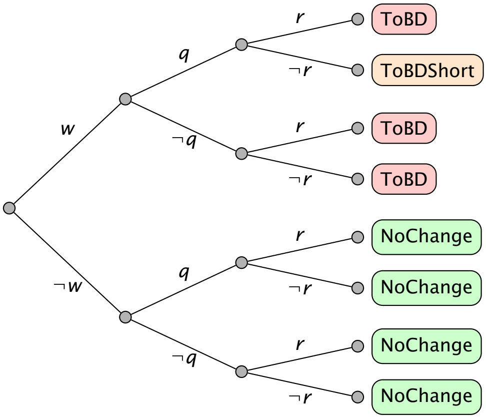
- **Traffic Lights outcome table**

w	q	r	Event
T	T	T	ToBD
T	T	F	ToBDShort
T	F	T	ToBD
T	F	F	ToBD
F	T	T	NoChange
F	T	F	NoChange
F	F	T	NoChange
F	F	F	NoChange

Boolean Expressions

Traffic Lights Example (3)

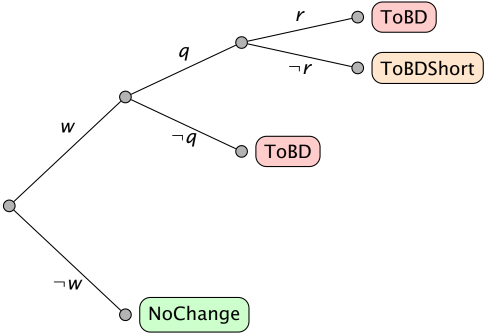
- **Traffic lights outcome tree**



Boolean Expressions

Traffic Lights Example (4)

- **Traffic lights outcome tree** simplified



Boolean Expressions

Traffic Lights Example (5)

- **Traffic Lights code 01**
- See M269TutorialProgPythonADT01.py

```
def trafficLights01(w,q,r) :
  """
  Input 3 Booleans
  Return Event string
  """
  if w :
    if q :
      if r :
        evnt = "ToBD"
      else :
        evnt = "ToBDShort"
    else :
      evnt = "ToBD"
  else :
    evnt = "NoChange"
  return evnt
```

Boolean Expressions

Traffic Lights Example (6)

- **Traffic Lights test code 01**

```
trafficLights01Evnts = [((w,q,r), trafficLights01(w,q,r))
                              for w in [True,False]
                              for q in [True,False]
                              for r in [True,False]]

assert trafficLights01Evnts \
  == [((True, True, True), 'ToBD')
       ,((True, True, False), 'ToBDShort')
       ,((True, False, True), 'ToBD')
       ,((True, False, False), 'ToBD')
       ,((False, True, True), 'NoChange')
       ,((False, True, False), 'NoChange')
       ,((False, False, True), 'NoChange')
       ,((False, False, False), 'NoChange')]
```

Boolean Expressions

Traffic Lights Example (7)

- **Traffic Lights code 02** compound Boolean conditions

```
def trafficLights02(w,q,r) :
  """
  Input 3 Booleans
  Return Event string
  """
  if ((w and q and r) or (w and not q)) :
    evnt = "ToBD"
  elif (w and q and not r) :
    evnt = "ToBDShort"
  else :
    evnt = "NoChange"
  return evnt
```

- What objectives do we have for our code ?

Boolean Expressions

Traffic Lights Example (8)

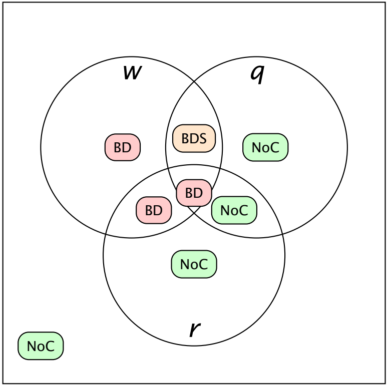
- **Traffic Lights test code 02**

```python
trafficLights02Evnts = [((w,q,r), trafficLights02(w,q,r))
                            for w in [True,False]
                            for q in [True,False]
                            for r in [True,False]]

assert trafficLights02Evnts == trafficLights01Evnts
```

Boolean Expressions

Traffic Lights Example (9)



- **Traffic Lights Venn diagram**
- OK using a fill colour would look better but didn't have the time to hack the package

Boolean Expressions

Validity

- **Validity** of Boolean expressions
- **Complete** every outcome returns an event (or error message, raises an exception)
- **Consistent** — we do not want two nested `if` statements or expressions resulting in different events
- We check this by ensuring that the events form a disjoint partition of the set of outcomes — see the Venn diagram
- We would quite like the programming language processor to warn us otherwise — not always possible

Booleans Expressions

Rail Ticket Exercise (1)

- Rail ticket discounts for:
  - *c* Rail card
  - *q* Off-peak time
  - *s* Special offer
- 4 fares: Standard, Reduced, Special, Super Special
- Rules:
  1. Reduced fare if rail card or at off-peak time
  2. Without rail card no reduction for both special offer and off-peak.
  3. Rail card always has reduced fare but cannot get off-peak discount as well.
  4. Rail card gets super special discount for journey with special offer
- Draw up truth table, outcome tree, Venn diagram and conditional statement (or expression) for this

Booleans Expressions

Rail Ticket Exercise (2)

- Rail ticket outcome table

c	q	s	Event
T	T	T	Super Special
T	T	F	Reduced
T	F	T	Super Special
T	F	F	Reduced
F	T	T	Special
F	T	F	Reduced
F	F	T	Special
F	F	F	Standard

Booleans Expressions

Rail Ticket Exercise (3)

- Rail ticket outcome table
- Note that it may be more convenient to change columns

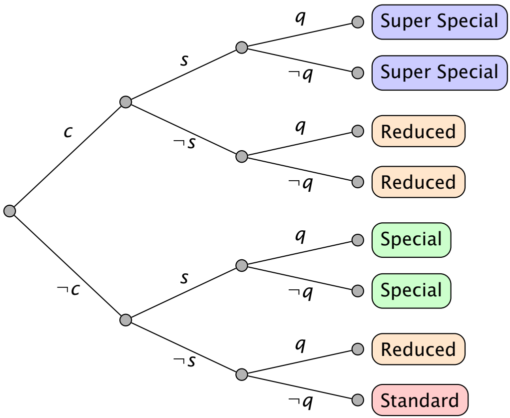
c	s	q	Event
T	T	T	Super Special
T	T	F	Super Special
T	F	T	Reduced
T	F	F	Reduced
F	T	T	Special
F	T	F	Special
F	F	T	Reduced
F	F	F	Standard

- Real fares are a little more complex — see brfares.com

Boolean Expressions

Rail Ticket Exercise (4)

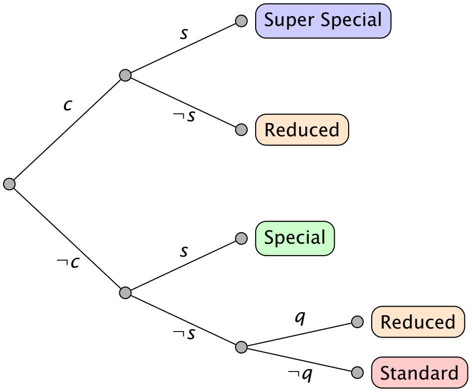
- **Rail Ticket outcome tree**



Boolean Expressions

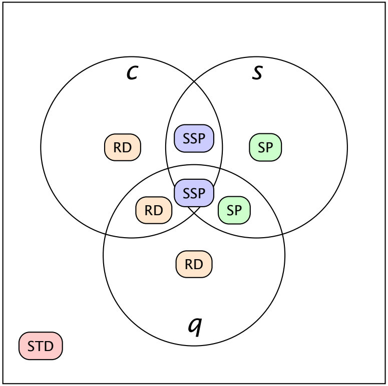
Rail Ticket Exercise (5)

- **Rail Ticket outcome tree** simplified



Boolean Expressions

Rail Ticket Example (6)



- **Rail Ticket Venn diagram**

Boolean Expressions

Rail Ticket Example (7)

- **Rail Ticket code 01**

```python
def railTicket01(c,s,q) :
  """
  Input 3 Booleans
  Return Event string
  """
  if c :
    if s :
      evnt = "SSP"
    else :
      evnt = "RD"
  else :
    if s :
      evnt = "SP"
    else :
      if q :
        evnt = "RD"
      else :
        evnt = "STD"
  return evnt
```

Boolean Expressions

Rail Ticket Example (8)

- **Rail Ticket test code 01**

```
railTicket01Evnts = [((c,s,q), railTicket01(c,s,q))
                              for c in [True,False]
                              for s in [True,False]
                              for q in [True,False]]

assert railTicket01Evnts \
  == [((True, True, True), 'SSP')
       ,((True, True, False), 'SSP')
       ,((True, False, True), 'RD')
       ,((True, False, False), 'RD')
       ,((False, True, True), 'SP')
       ,((False, True, False), 'SP')
       ,((False, False, True), 'RD')
       ,((False, False, False), 'STD')]
```

Boolean Expressions

Rail Ticket Example (9)

- **Rail Ticket code 02** compound Boolean expressions

```
def railTicket02(c,s,q) :
  """
  Input 3 Booleans
  Return Event string
  """
  if (c and s) :
    evnt = "SSP"
  elif ((c and not s) or (not c and not s and q)) :
    evnt = "RD"
  elif (not c and s) :
    evnt = "SP"
  else :
    evnt = "STD"
  return evnt
```

Boolean Expressions

Rail Ticket Example (10)

- **Rail Ticket test code 02**

```
railTicket02Evnts = [((c,s,q), railTicket02(c,s,q))
                                for c in [True,False]
                                for s in [True,False]
                                for q in [True,False]]

assert railTicket02Evnts == railTicket01Evnts
```

Propositional Calculus

Introduction

- Unit 2 section 3.2 *A taste of formal logic* introduces Propositional calculus
- A language for calculating about Booleans — *truth values*
- Gives operators (connectives) conjunction ($\wedge$) AND, disjunction ($\vee$) OR, negation ($\neg$) NOT, implication ($\Rightarrow$) IF
- There are 16 possible functions $(\mathbb{B}, \mathbb{B}) \rightarrow \mathbb{B}$ — see below — defined by their truth tables
- **Discussion** Did you find the truth table for implication weird or surprising ?

Propositional Calculus

Implication

- **Implication** has a **negative** definition — we accept its truth unless we have contrary evidence
- $T \Rightarrow T == T$ and $T \Rightarrow F == F$
- Hence 4 possibilities for truth table

p	q	$p \Rightarrow q$	q	$p \Leftrightarrow q$	$p \wedge q$
T	T	T	T	T	T
T	F	F	F	F	F
F	T	T	T	F	F
F	F	T	F	T	F

- $(\Rightarrow)$ must have the entry shown — the others are taken
- Do not think of *p causing q*

Propositional Calculus

Functional Completeness, Boolean Programming

- **Functionally complete** set of connectives is one which can be used to express all possible connectives
- $p \Rightarrow q \equiv \neg p \vee q$ so we could just use $\{\neg, \wedge, \vee\}$
- **Boolean programming** — we have to have a functionally complete set but choose more to make the programming easier
- *Expressiveness* is an issue in programming language design

Propositional Calculus

NAND, NOR

- **NAND** $\boldsymbol{p \overline{\wedge} q}$, $\boldsymbol{p \uparrow q}$, Sheffer stroke
- **NOR** $\boldsymbol{p \overline{\vee} q}$, $\boldsymbol{p \downarrow q}$, Pierce's arrow
- See truth tables below — both $\{\uparrow\}, \{\downarrow\}$ are functionally complete
- **Exercise** verify
  - $\neg p \equiv p \uparrow p$
  - $p \wedge q \equiv \neg(p \uparrow q) = (p \uparrow q) \uparrow (p \uparrow q)$
  - $p \vee q \equiv (p \uparrow p) \uparrow (q \uparrow q)$
  - $\neg p \equiv p \downarrow p$
  - $p \wedge q \equiv (p \downarrow p) \downarrow (q \downarrow q)$
  - $p \vee q \equiv \neg(p \downarrow q) = (p \downarrow q) \downarrow (p \downarrow q)$
- Not a novelty — the Apollo Guidance Computer was implemented in NOR gates alone.

Truth Function

Truth Function References

- The following appendix notes illustrate the 16 binary functions of two Boolean variables
- See Truth function
- See Functional completeness
- See Sheffer stroke
- See Logical NOR

Truth Function

Table of Binary Truth Functions

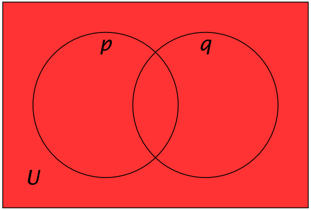
p	q	$\top$	$p \lor q$	$p \Leftarrow q$	p	$p \Rightarrow q$	q	$p \Leftrightarrow q$	$p \land q$
T	T	T	T	T	T	T	T	T	T
T	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F

p	q	$\bot$	$p \overline{\lor} q$	$p \nLeftarrow q$	$\neg p$	$p \nRightarrow q$	$\neg q$	$p \nLeftrightarrow q$	$p \overline{\land} q$
T	T	F	F	F	F	F	F	F	F
T	F	F	F	F	F	T	T	T	T
F	T	F	F	T	T	F	F	T	T
F	F	F	T	F	T	F	T	F	T

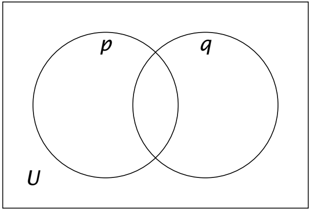
Truth Function

Tautology/Contradiction

- **Tautology** True, $\top$, *Top*



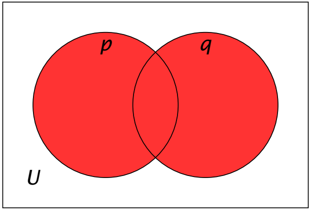
- **Contradiction** False, $\perp$, *Bottom*



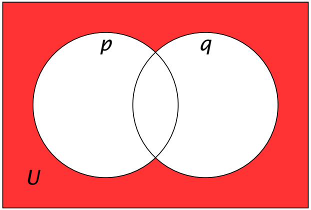
Truth Function

Disjunction/Joint Denial

- **Disjunction** OR, $p \vee q$



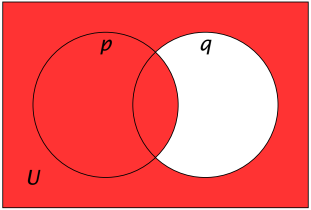
- **Joint Denial** NOR, $p \overline{\vee} q$, $p \downarrow q$, *Pierce's arrow*



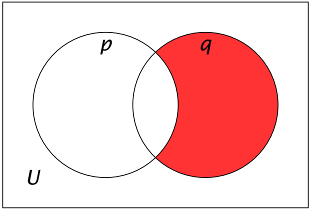
Truth Function

Converse Implication/Converse Nonimplication

- **Converse Implication** $p \Leftarrow q$



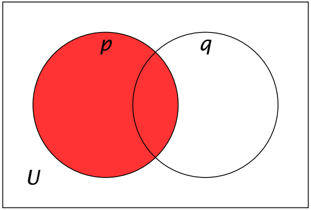
- **Converse Nonimplication** $p \not\Leftarrow q$



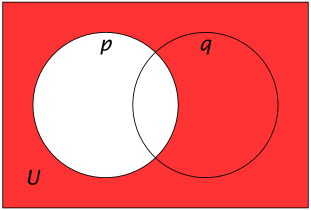
Truth Function

Proposition p/Negation of p

- **Proposition** p



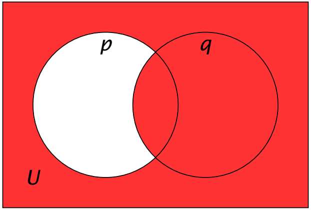
- **Negation of** p



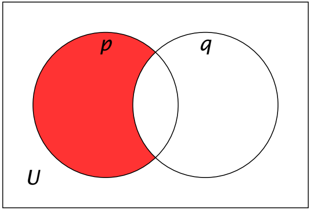
Truth Function

Material Implication/Material Nonimplication

- **Material Implication** $p \Rightarrow q$



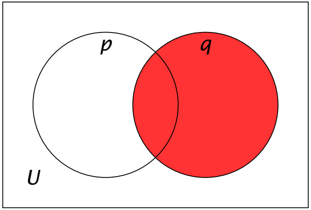
- **Material Nonimplication** $p \nRightarrow q$



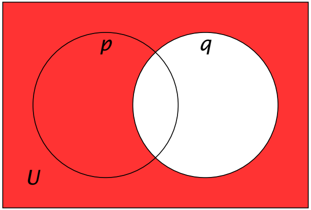
Truth Function

Proposition q/Negation of q

- **Proposition** q q



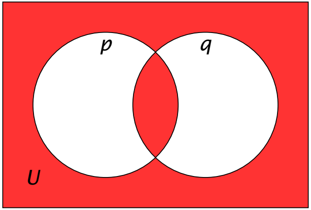
- **Negation of** q $\neg q$



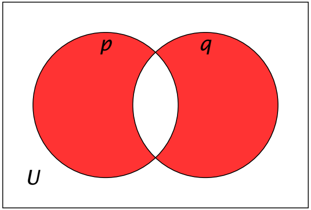
Truth Function

Biconditional/Exclusive disjunction

- **Biconditional** If and only if, IFF, $p \Leftrightarrow q$



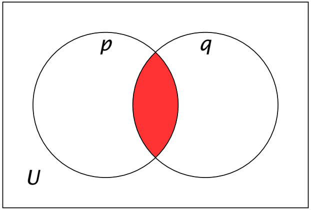
- **Exclusive disjunction** XOR, $p \not\Leftrightarrow q$



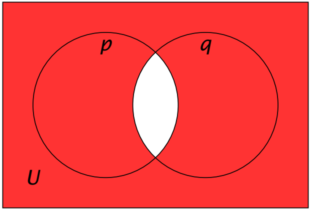
Truth Function

Conjunction/Alternative denial

- **Conjunction** AND, $p \wedge q$



- **Alternative denial** NAND, $p \barwedge q$, $p \uparrow q$, *Sheffer stroke*



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 `Circle`s, `Square`s, . . . 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

```haskell
data Shape
  = Circle Point Radius
   | Square Point Size

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

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

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

shapes01 :: [Shape]
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

```haskell
class IsShape shape where
  draw :: shape -> Pict
  moveTo :: Point -> shape -> shape

data Shape = forall aTVar . (IsShape aTVar) => Shape aTVar

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

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

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

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

Haskell Code

Commentary (1)

- The following is a *very* brief commentary on the Haskell code

```
data Shape
  = Circle Point Radius
   | Square Point Size
```

- **data** defines an algebraic datatype with two constructors (`Circle`, `Square`) which each take two arguments of types assumed to be defined elsewhere (*`Point`*, *`Radius`*, *`Size`*)

Haskell Code

Commentary (2)

```haskell
draw :: Shape -> Pict
draw (Circle p r) = drawCircle p r
draw (Square p s) = drawRectangle p s s

moveTo :: Point -> Shape -> Shape
moveTo p2 (Circle p1 r) = Circle p2 r
moveTo p2 (Square p1 s) = Square p2 s
```

- The lines starting `draw ::` and `moveTo ::` are *type signatures* which specify types for the functions `draw` and `moveTo`
- In each case the next couple of lines define the function
- Note that function and constructor application is denoted by juxtaposition, is left associative and is more binding than (almost) anything else
- `(f x y)` means `((f x) y)`

Haskell Code

Commentary (3)

```haskell
class IsShape shape where
  draw :: shape -> Pict
  moveTo :: Point -> shape -> shape
```

- The above declares the type class IsShape which includes the type signatures of the functions which must be defined in any instance declaration

```haskell
instance IsShape Circle where
  draw (Circle p r) = drawCircle p r
  moveTo p2 (Circle p1 r) = Circle p2 r
```

- The above is an instance declaration for the type Circle to be a member of the type class IsShape

Haskell Code

Commentary (3)

```haskell
data Shape = forall aTVar . (IsShape aTVar) => Shape aTVar
```

- The above declares Shape to have the constructor Shape which takes a type variable *aTVar* which is a member of the type class `IsShape`
- See What does the `forall` keyword do ?
- Understanding **`forall`** requires some knowledge of first-order logic so initially may appear a bit subtle.
- Norman Ramsey in the above StackOverflow article recommends John Launchbury and Simon Peyton Jones (1994) *Lazy Functional State Threads*
- Also see HaskellWiki: Existential type
- Note that in Haskell reserved words and variable names (including functions) and type variables start with a lower case letter while names for particular values or types start with an upper case letter (with a few exceptions)

Abstract Data Types

ADT/OO Colour Codes

- **Default: Language Main Constructs**

  data map class where forall instance (Haskell)
- **Language Builtin other**

  upper find count (Python)
- **User Defined**

  Shape Circle Square IsShape draw moveTo (Haskell)
- **Meta**

  decls, decl1, decl2, declK, expr, alts
- **Special**

  GHCi> (Haskell GHCi)
- **Meta Builtin**

  shape aTVar type variables (Haskell)
- **Meta User Defined**

  Pict Point Radius drawCircle drawRectangle Haskell

Abstract Data Types

Overview (6) — The Expression Problem

- The *Expression Problem* describes a dual problem that neither Object Oriented Programming nor Functional Programming fully addresses.
- If you want to add a new thing, Object Oriented Programming makes it easy (since you can simply create a new class) but Functional Programming makes it harder (since you have to edit every function that accepts a thing of that type)
- If you want to add a new function, Functional Programming makes it easy (simply add a new function) while Object Oriented Programming makes it harder (since you have to edit every class to add the function)
- Wikipedia: Expression problem
- Bendersky: The Expression Problem and More thoughts
- C2 Wiki: Expression Problem
- What is the 'expression problem'?
- Philip Wadler: The Expression Problem

Abstract Data Type

Queue

- Queue Abstract Data Type — operations
- `makeEmptyQ` returns empty queue
- `isEmptyQ` takes queue, returns Boolean
- `addToQ` takes queue, item, returns queue with item added at back
- `headOfQ` takes queue, returns item at front
- `tailOfQ` takes queue, returns queue without front item
- Other operations
- `removeFrontQ` takes queue, returns pair of item on the front and queue with item removed
- `sizeQ` to save calculating it
- `isFullQ` for a bounded queue

Abstract Data Type

Queue (2)

- **Pre, Post Conditions, Axioms** should be **complete**
- They define all permissable inputs to the functions (or methods)
- They define the outcome of all applications of the functions
- Composition of the functions constructs all possible members of the ADT set

Abstract Data Type

Queue (3) — Pre-conditions, Post-conditions, Axioms

- **Pre-conditions, Post-conditions, Axioms**
- `makeEmptyQ()`
- **Pre** `True`
- **Post** Return value `q` is an empty queue
- **Axiom** `makeEmptyQ() == EmptyQ`
- `isEmptyQ()`
- **Pre** `True`
- **Post** Returns `True` if `q` is empty, otherwise `False`
- **Axiom** `isEmptyQ(makeEmptyQ()) == True`
- `isEmptyQ(addToQ(q,x)) == False`
- **Exercise** complete this for the other operations

Abstract Data Type

Queue (4) — Pre-conditions, Post-conditions, Axioms

- **Pre-conditions, Post-conditions, Axioms**
- `addToQ()`
- **Pre** `True`
- **Post** Returns queue with `x` at back, front part is input queue
- `headOfQ()`
- **Pre** Argument `q` is non-empty
- **Post** Return value is item at the front (queue is unchanged)
- **Axioms** `headOfQ(makeEmptyQ()) == error`
- `headOfQ(addToQ(makeEmptyQ(),x)) == x`
- `headOfQ(addToQ(q,x)) == headOfQ(q)`

Abstract Data Type

Queue (5) — Pre-conditions, Post-conditions, Axioms

- **Pre-conditions, Post-conditions, Axioms**
- `tailOfQ()`
- **Pre** `True`
- **Post** Returns queue without first item
- **Axioms** `tailOfQ(makeEmptyQ()) == error`
- `tailOfQ(addToQ(makeEmptyQ(),x)) == EmptyQ`
- `tailOfQ(addToQ(q,x)) == addToQ(tailOfQ(q),x)`

Abstract Data Type

Queue Implementation (1)

- **Queue Implementation**
- Using Lists as Queues section 5.1.2 of the *Tutorial*
- Quote: It is also possible to use a list as a queue, where the first element added is the first element retrieved (first-in, first-out); however, lists are not efficient for this purpose. While appends and pops from the end of list are fast, doing inserts or pops from the beginning of a list is slow (because all of the other elements have to be shifted by one).
- Could use `collections.deque` but we will use a pair of lists — See Okasaki (1998, page 42)

Abstract Data Type

Queue Implementation (2)

- **Queue Implementation 1**
- Using a `namedtuple()`
- A factory function for creating tuple subclasses with named fields

```
from collections import namedtuple

Qp1 = namedtuple('Qp1',['frs','rbks'])
```

Abstract Data Type

Queue Implementation (3)

- **Queue Implementation 1** main operations

```
def makeEmptyQp1():
  return Qp1([],[])

def isEmptyQp1(q):
  return q.frs == []

def addToQp1(q,x):
  return checkQp1(q.frs, [x] + q.rbks[:])

def headOfQp1(q):
  if q.frs == [] :
    RuntimeError("headOfQp1␣applied␣to␣empty␣queue")
  else:
    return q.frs[0]

def tailOfQp1(q):
  if q.frs == [] :
    RuntimeError("tailOfQp1␣applied␣to␣empty␣queue")
  else:
    return checkQp1(q.frs[1:], q.rbks[:])
```

Abstract Data Type

Queue Implementation (4)

- **Queue Implementation 1** checkQp1()

```python
def checkQp1(frs, rbks):
  if frs == [] :
    bks = rbks[:]
    bks.reverse()
    return Qp1(bks, [])
  else :
    return Qp1(frs, rbks)
```

- Note copying of arguments — see below for reason
- Note also in stringQp1Items below at line 47 on slide 194
- implicit line joining using (()) (why is this needed ??)
- Note use of recursion

Abstract Data Type

Python Argument Passing

- **Functions, Immutable and Mutable Arguments**
- Immutable arguments are passed by value
- Mutable arguments are passed by reference
- Immutable: numbers, strings, tuples
- Mutable: Lists, dictionaries, sets, and most objects in user classes

```
>>> def changer (a,b) :
...   a = 2
...   b[0] = 'spam'
...
>>> n = 1
>>> xs = [1,2]
>>> changer(n, xs)
>>> (n,xs)
(1, ['spam', 2])
```

Abstract Data Type

Queue Implementation (5)

- **Queue Implementation 1** conversion operations

```python
def stringQp1(q) :
  return ("<" + stringQp1Items(q) + ">")

def stringQp1Items(q) :
  if isEmptyQp1(q) :
    return ""
  elif isEmptyQp1(tailOfQp1(q)) :
    return str(headOfQp1(q))
  else :
    return ( str(headOfQp1(q))
            + ", " + stringQp1Items(tailOfQp1(q)) )

def buildQp1(xs,q) :
  if xs == [] :
    return q
  else :
    return buildQp1(xs[1:],addToQp1(q,xs[0]))

def listToQp1(xs) :
  return buildQp1(xs, makeEmptyQp1())
```

Abstract Data Type

Queue Implementation (6)

- **Queue Implementation 1** test code

```python
q11 = listToQp1([1,2,3,1])

q12 = tailOfQp1(q11)

assert q11 == Qp1(frs=[1], rbks=[1, 3, 2])

assert stringQp1(q11) == '<1, 2, 3, 1>'

assert q12 == Qp1(frs=[2, 3, 1], rbks=[])

assert stringQp1(q12) == '<2, 3, 1>'
```

Abstract Data Type

Queue Implementation (7)

- **Queue Implementation 2**
- Modify to add size
- Store in tuple to save calculating each time

```
Qp2 = namedtuple('Qp2',['frs','rbks','sz'])
```

- **Exercise** Add size() operation and other modifications

Abstract Data Type

Queue Implementation (8)

- **Queue Implementation 2** main operations

```python
def makeEmptyQp2():
  return Qp2([],[], 0)

def isEmptyQp2(q):
  return q.frs == []

def addToQp2(q,x):
  return checkQp2(q.frs, [x] + q.rbks[:], q.sz + 1)

def headOfQp2(q):
  if q.frs == [] :
    RuntimeError("headOfQp2␣applied␣to␣empty␣queue")
  else:
    return q.frs[0]

def tailOfQp2(q):
  if q.frs == [] :
    RuntimeError("tailOfQp2␣applied␣to␣empty␣queue")
  else:
    return checkQp2(q.frs[1:], q.rbks[:], q.sz - 1)
```

Abstract Data Type

Queue Implementation (9)

- **Queue Implementation 2** sizeQp2(), checkOp1()

```python
def sizeOfQp2(q) :
  return q.sz

def checkQp2(frs, rbks, sz):
  if frs == [] :
    bks = rbks[:]
    bks.reverse()
    return Qp2(bks, [], sz)
  else :
    return Qp2(frs, rbks, sz)
```

- Note also in stringQp2Items below at line 118 on slide 199
- implicit line joining using (()) (why is this needed ??)
- Note use of recursion

Abstract Data Type

Queue Implementation (10)

- **Queue Implementation 2** conversion operations

```
def stringQp2(q) :
  return ("<" + stringQp2Items(q) + ">")

def stringQp2Items(q) :
  if isEmptyQp2(q) :
    return ""
  elif isEmptyQp2(tailOfQp2(q)) :
    return str(headOfQp2(q))
  else :
    return ( str(headOfQp2(q))
             + ", " + stringQp2Items(tailOfQp2(q)) )

def buildQp2(xs,q) :
  if xs == [] :
    return q
  else :
    return buildQp2(xs[1:],addToQp2(q,xs[0]))

def listToQp2(xs) :
  return buildQp2(xs, makeEmptyQp2())
```

Abstract Data Type

Queue Implementation (11)

- **Queue Implementation 2** test code

```python
q21 = listToQp2([1,2,3,1])

q22 = tailOfQp2(q21)

assert q21 == Qp2(frs=[1], rbks=[1, 3, 2], sz=4)

assert stringQp2(q21) == '<1, 2, 3, 1>'

assert q22 == Qp2(frs=[2, 3, 1], rbks=[], sz=3)

assert stringQp2(q22) == '<2, 3, 1>'
```

Abstract Data Types

Lists Implemented in Lists (1)

- Lists implemented naively as linked lists have some operations that take constant time and some that are linear in the length of the list
- Adding an element to the front of a list takes constant time while adding an element to the rear takes linear time
- This section reimplements lists using a pair of lists that overcomes this asymmetry in efficiency giving constant time for all operations.
- The basic idea is quite simple: break the list in two and reverse the second half
- This means that the last element is the first element of the second list
- A problem arises when one attempts to remove an element — in some cases the list has to be reorganised into two halves
- The criteria for reorganising gives the clue in how to write the code

Abstract Data Types

Lists Implemented in Lists (2)

- This implementation is based on Bird and Gibbons (2020, chp 3) *Algorithm Design with Haskell*
- The idea is attributed to Gries (1981, page 250) *The Science of Programming* and Hood and Melville (1980) *Real time queue operations in pure Lisp*
- See also Hoogerwoord (1992) *Functional Pearls A symmetric set of efficient list operations*

Abstract Data Types

Lists Implemented in Lists (3)

- We give the code in Python from SymmetricLists.py with Haskell type specifications and declarations given as comments
- Here is the type alias declaration as a comment along with fromSL which converts back from symmetric lists to standard lists — this is known as the *abstraction function*

```python
#  type SymList a = ([a],[a])

# Abstraction function

# fromSL :: SymList a -> [a]

def fromSL (pr) :
  xs = pr[0]
  ys = pr[1]
  return xs + reverseF (ys)

def reverseF (xs) :
  ys = xs[:]
  ys.reverse()
  return ys
```

Abstract Data Types

Lists Implemented in Lists (4)

- The `abstraction function` captures the relationship between the implementation of an operation on the representing type and its abstract type with an equation
- The *Eureka* bit of the implementation is spotting the *representation invariant* that our definitions both exploit and maintain

```
# repInvSL :: SymList a -> Bool

def repInvSL (pr) :
  xs = pr[0]
  ys = pr[1]
  xsTest = ((not isEmpty (xs))
            or (isEmpty (ys) or singleton (ys)))
  ysTest = ((not isEmpty (ys))
            or (isEmpty (xs) or singleton (xs)))
  return (xsTest and ysTest)
```

- This says if one list is empty then the other must be either empty or a singleton
- This tells us when we need to reorganise the lists

Abstract Data Types

Lists Implemented in Lists (5)

- Here are the service operations for empty lists and singletons

```
# isEmpty :: [a] -> Bool

def isEmpty (xs) :
  return (xs == [])

# isEmptySL :: SymList a -> Bool

def isEmptySL (pr) :
  xs = pr[0]
  ys = pr[1]
  return (isEmpty (xs) and isEmpty (ys))

def makeEmptySL() :
  return ([],[])
```

Abstract Data Types

Lists Implemented in Lists (6)

```
# singleton :: [a] -> Bool

def singleton (xs) :
  return (len(xs) == 1)

# singletonSL :: SymList a -> Bool

def singletonSL (pr) :
  xs = pr[0]
  ys = pr[1]
  return ((isEmpty (xs) and singleton (ys))
          or (isEmpty (ys) and singleton (xs)))
```

Abstract Data Types

Lists Implemented in Lists (7)

- Constructor operations
- Both of these definitions make use of the representation invariant

```
#  Constructor functions

# consSL :: a -> SymList a -> SymList a

def consSL (x, pr) :
  xs = pr[0]
  ys = pr[1]
  if isEmpty (ys) :
    return ([x],xs)
  else :
    return ([x] + xs, ys)

# snocSL :: a -> SymList a -> SymList a

def snocSL (x, pr) :
  xs = pr[0]
  ys = pr[1]
  if isEmpty (xs) :
    return (ys,[x])
  else :
    return (xs, [x] + ys)
```

Abstract Data Types

Lists Implemented in Lists (8)

- Inspectors

```
# headSL :: SymList a -> a

def headSL (pr) :
  xs = pr[0]
  ys = pr[1]
  if isEmpty (xs) :
    if isEmpty (ys) :
      raise RuntimeError("headSL ([],[])")
    else :
      return ys[0]
  else :
    return xs[0]

# lastSL :: SymList a -> a

def lastSL (pr) :
  xs = pr[0]
  ys = pr[1]
  if isEmpty (ys) :
    if isEmpty (xs) :
      raise RuntimeError("tailSL ([],[])")
    else :
      return xs[0]
  else :
    return ys[0]
```

Abstract Data Types

Lists Implemented in Lists (9)

- `tailSL`
- Notice how the representation invariant is maintained

```
# tailSL :: SymList a -> SymList a

def tailSL (pr) :
  xs = pr[0]
  ys = pr[1]
  if isEmpty (xs) :
    if isEmpty (ys):
      raise RuntimeError("tailSL ([],[])")
    else:
      return ([],[])
  elif singleton (xs) :
    splitPt = len(ys) // 2
    (us,vs) = (ys[:splitPt],ys[splitPt:])
    return (reverseF (vs), us)
  else :
    return (xs[1:],ys)
```

Abstract Data Types

Lists Implemented in Lists (10)

- initSL

```
# initSL :: SymList a -> SymList a

def initSL (pr) :
  xs = pr[0]
  ys = pr[1]
  if isEmpty (ys) :
    if isEmpty (xs):
      raise RuntimeError("initSL␣([],[])")
    else:
      return ([],[])
  elif singleton (ys) :
    splitPt = len(xs) // 2
    (us,vs) = (xs[:splitPt],xs[splitPt:])
    return (us, reverseF (vs))
  else :
    return (xs,ys[1:])
```

Abstract Data Types

Lists Implemented in Lists (11)

- The implementations are designed to satisfy the six equations:
- The equations are expressed here in Haskell notation

```
# -- The implementation satisfies the following
# --
# -- (cons x . fromSL) ps == (fromSL . consSL x) ps
# -- (snoc x . fromSL) ps == (fromSL . snocSL x) ps
# -- (tail . fromSL) ps   == (fromSL . tailSL) ps
# -- (init . fromSL) ps   == (fromSL . initSL) ps
# -- (head . fromSL) ps   == headSL ps
# -- (last . fromSL) ps   == lastSL ps
```

- Each of the operations apart from `tailSL` and `initSL` take constant time
- `tailSL` and `initSL` can take linear time in the worst case but they take amortised constant time — see the references for derivation
- Note that Haskell `Data.Sequence` uses 2-3 Finger Trees for better performance

Abstract Data Types

Lists Implemented in Lists (12)

- **Ex (1)** Write down all the ways "abcd" can be represented as a symmetric list.

  Give examples to show how each of these representations can be generated.
- **Ex (2)** Define `lengthSL`
- **Ex (3)** Implement `dropWhileSL` so that

```
# dropWhile . fromSL = fromSL . dropWhileSL
```

- **Ex (4)** Define `initsSL` with the type

```
# initsSL :: SymList a -> SymList (SymList a)
```

  Write down the equation which expresses the relationship between `fromSL`, `initsSL`, and `inits`.
- Note Exs (3) and (4) use Python beyond M269

Abstract Data Types

Lists Implemented in Lists (12a)

- **Ans (1)** There are three ways:

```
("a","dcb"),("ab","dc"),("abc","d")
```

```
Python3>>> prs1 = consSL('a',([],[]))
Python3>>> prs1
(['a'], [])
Python3>>> prs2 = snocSL('b',prs1)
Python3>>> prs2
(['a'], ['b'])
Python3>>> prs3 = snocSL('c',prs2)
Python3>>> prs3
(['a'], ['c', 'b'])
Python3>>> prs4 = snocSL('d',prs3)
Python3>>> prs4
(['a'], ['d', 'c', 'b'])

Python3>>> prs1a = snocSL('a',([],[]))
Python3>>> prs1a
([], ['a'])
Python3>>> prs2a = snocSL('b',prs1a)
Python3>>> prs2a
(['a'], ['b'])
```

Abstract Data Types

Lists Implemented in Lists (12b)

- **Ans (1)** There are three ways:

```
("a","dcb"),("ab","dc"),("abc","d")
```

```
Python3>>> prs1 = consSL('d',([],[]))
Python3>>> prs1
(['d'], [])
Python3>>> prs2 = consSL('c',prs1)
Python3>>> prs2
(['c'], ['d'])
Python3>>> prs3 = consSL('b',prs2)
Python3>>> prs3
(['b', 'c'], ['d'])
Python3>>> prs4 = consSL('a',prs3)
Python3>>> prs4
(['a', 'b', 'c'], ['d'])
```

- Functional programmers will spot that the first is an instance of a `foldl` while the third is an instance of a `foldr`

Abstract Data Types

Lists Implemented in Lists (12c)

- **Ans (2)** Define `lengthSL`

```
# lengthSL :: SymList -> Int

def lengthSL (pr) :
xs = pr[0]
ys = pr[1]
return len(xs) + len(ys)
```

- Note that we have made `lengthSL` a function in the SymmetricList ADT that has access to the underlying implementation
- We could have done that without giving it access but that would have a cost
- Since `lengthSL` is used a lot we give it access

Abstract Data Types

Lists Implemented in Lists (12d)

- **Ans (3)** Implement dropWhileSL

```
def dropWhileSL(pred, xs) :
  if isEmptySL(xs) :
    return makeEmptySL
  elif pred(headSL(xs)) :
    return dropWhileSL(pred, (tailSL(xs)))
  else :
    return xs
```

- Note pred is a function object, defining a function that takes one argument and returns a Boolean
- dropWhileSL is an example of a *higher order function* — a function that can take or receive a function as an argument

Abstract Data Types

Lists Implemented in Lists (12d)

- **Ans (3)** Test code

```
def greaterThan5 (x) :
  return x > 5

testList3 = [25, 35, 4, 45]
testList3SL = toSL(testList3)

test3A = (testList3SL == ([25, 35], [45, 4]))
test3B = (fromSL(dropWhileSL(greaterThan5,testList3SL)) == [4, 45])
```

Abstract Data Types

Lists Implemented in Lists (12e)

- **Ans (4)** Implement initsSL(xs)

```
def initsSL(xs) :
  if isEmptySL(xs) :
    return (snocSL(xs, makeEmptySL()))
  else :
    return (snocSL(xs, (initsSL(initSL(xs)))))
```

Abstract Data Types

Lists Implemented in Lists (12f)

- **Ans (4)** Test code

```
test4inits = initsSL(testList3SL)

test4initsFromSL = fromSL (test4inits)

def fromSLs(prs) :
  if prs == [] :
    return []
  else :
    return [fromSL(prs[0])] + fromSLs(prs[1:])

def displayInitsSL(xs) :
  return fromSLs(fromSL(initsSL(xs)))

test4Display = displayInitsSL(testList3SL)
```

```
Python3>>> testList3
[25, 35, 4, 45]
Python3>>> testList3SL
([25, 35], [45, 4])
Python3>>> test4Display
[[], [25], [25, 35], [25, 35, 4], [25, 35, 4, 45]]
```

Abstract Data Types

Lists Implemented in Lists (12g)

- **Ans (4)** Relationship between `fromSL`, `initsSL`, and `inits`.
- In Haskell first, followed by Python
- Why that way round ? Haskell makes it more obvious what is going on

```
(inits . fromSL) prs = (map fromSL . fromSL . initsSL) prs
```

- `(.)` is the Haskell function composition operator

```
inits(fromSL(prs)) = map(fromSL, fromSL(initsSL(prs)))
```

- map in Python does (roughly) the same as Haskell map
- However, in Python map returns an iterator, which represents a stream of data
- This means we need some extra code to print the result — maybe using the type constructor `list(`*`iterable`*`)` or the alternative `fromSLs` and `displayInitsSL`

Abstract Data Types

Lists Implemented in Lists (12h)

- **Ans (4)** Using a list comprehension instead of an explicit map

```
def displayInitsSL01(symList) :
  return [fromSL(pr) for pr in fromSL(initsSL(symList))]
```

What Next ?

Programming, Debugging, Psychology

Although programming techniques have improved immensely since the early days, the process of finding and correcting errors in programming — known graphically if inelegantly as *debugging* — still remains a most difficult, confused and unsatisfactory operation. The chief impact of this state of affairs is psychological. Although we are happy to pay lip-service to the adage that to err is human, most of us like to make a small private reservation about our own performance on special occasions when we really try. It is somewhat deflating to be shown publicly and incontrovertibly by a machine that even when we do try, we in fact make just as many mistakes as other people. If your pride cannot recover from this blow, you will never make a programmer.

Christopher Strachey, Scientific American 1966 vol 215 (3) September pp112–124

What Next ?

To err is human ?

- To err is human, to really foul things up requires a computer.
- Attributed to Paul R. Ehrlich in 101 Great Programming Quotes
- Attributed to Bill Vaughn in Quote Investigator
- Derived from Alexander Pope (1711, An Essay on Criticism)
- *To Err is Humane; to Forgive, Divine*
- This also contains

  A little learning is a dangerous thing;
  Drink deep, or taste not the Pierian Spring

- In programming, this means you have to *read the fabulous manual* (RTFM)

Future Work

Sorting, Searching

- Recursive function definitions
- Inductive data type definitions
  - A *list* is either an empty list or a first item followed by the rest of the list
  - A *binary tree* is either an empty tree or a node with an item and two sub-trees
- Recursive definitions often easier to find than iterative
- Sorting
- Searching
- Both use binary tree structure

Future Work

Dates

- TMA01 Thursday 19 December 2024 TMA01
- Sunday 5 January 2025 Tutorial Online Sorting
- Sunday 12 January 2025 Tutorial Online Binary Trees
- Sunday 9 February 2025 Tutorial Online Binary Trees
- TMA02 Thursday 6 March 2025 TMA02

Example Algorithm Design — Haskell

Binary Search — Haskell

- The notes following give two implementations of *Binary Search* in Haskell
- Note: these are not part of M269 and are purely for comparison for those interested
- The first is a direct translation of the recursive Python version
- The second is derived from `http://rosettacode.org/wiki/Binary_search` and is more idiomatic Haskell
- The code for both implementations is in the file `M269BinarySearch.hs` (which should be near the file of these slides)

Example Algorithm Design — Haskell

Binary Search — Haskell — 1 (a)

```haskell
module M269BinarySearch where

  import Data.Array
  import Data.List
```

- A Haskell script starts with a module header which starts with the reserved identifier, `module` followed by the module name, `M269BinarySearch`
- The module name must start with an upper case letter and is the same as the file name (without its extension of *.hs* or *.lhs*)
- Haskell uses *layout* (or the *off-side rule*) to determine scope of definitions, similar to Python
- The body of the module follows the reserved identifier `where` and starts with `import` declarations
- This imports the libraries `Data.List`, `Data.Array`

Example Algorithm Design — Haskell

Binary Search — Haskell — 1 (b)

```
6   binarySearch :: Ord a => [a] -> a -> Maybe Int

8   binarySearch xs val
9     = binarySearch01 xs val (lo,hi)
10      where
11      lo = 0
12      hi = length xs - 1
```

- Line 8 is the definition of `binarySearch`
- The preceding line, 6, is the *type signature*
- `binarySearch` takes a list and a value of type `a` (in the class `Ord` for ordering) and returns a `Maybe Int` — `a` is a *type variable*
- The `Maybe a` type is an *algebraic data type* which is the union of the *data constructors* `Nothing` and `Just a`

```
data Maybe a = Nothing | Just a
```

Example Algorithm Design — Haskell

Code Description 1

- `f :: t` is a *type signature* for variable `f` that reads *`f` is of type `t`*
- `f :: t1 -> t2` means that `f` has the type of a function that takes elements of type `t1` and returns elements of type `t2`
- The *function type arrow* `->` associates to the right
  - `f :: t1 -> t2 -> t3` means
  - `f :: t1 -> (t2 -> t3)`
- `f x` — function application is denoted by juxtaposition and is more binding than (almost) any other operation.
- *Function application* is left associative
  - `f x y` means
  - `(f x) y`

Example Algorithm Design — Haskell

Binary Search — Haskell — 1 (c)

```
binarySearch01 :: Ord a
  => [a] -> a -> (Int, Int) -> Maybe Int

binarySearch01 xs val (lo,hi)
  = if  hi < lo then Nothing
    else
      let mid = (lo + hi) `div` 2
          guess = xs !! mid
      in
      if val == guess
        then Just mid
      else if val < guess
        then binarySearch01 xs val (lo,mid-1)
      else   binarySearch01 xs val (mid + 1, hi)
```

Example Algorithm Design — Haskell

Code Description 2

- A `let` expression has the form

```
let decls in expr
```

- *decls* is a number of *declarations*
- *expr* is an expression (which is the scope of the declarations)
- `div` is the integer division function
- In `` `div` ``, the grave accents (`` ` ``) make a function into an infix operator (OK, that is syntactic sugar I need not have introduced — and my formatting program has coerced the grave accent to a left single quotation mark Unicode U+2018, not the grave accent U+0060)
- `(!!)` is the list index operator — first item has index 0

Example Algorithm Design — Haskell

Binary Search — Haskell — 2 (a)

```haskell
binarySearchGen :: Integral a
  => (a -> Ordering) -> (a, a) -> Maybe a
binarySearchGen p (lo,hi)
  | hi < lo = Nothing
  | otherwise =
    let mid = (lo + hi) `div` 2 in
    case p mid of
      LT -> binarySearchGen p (lo, mid - 1)
      GT -> binarySearchGen p (mid + 1, hi)
      EQ -> Just mid
```

Example Algorithm Design — Haskell

Code Description 3

- A `case` expression has the form

```
case expr of alts
```

  expr is evaluated and whichever alternative of *alts* matches is the result

- The lines starting with `(|)` are *guarded* definitions — if the boolean expression to the right is `True` then the following expression is used
- `otherwise` is a synonym for `True`
- A *conditional* expression has the form

```
if expr then expr else expr
```

  The first *expr* must be of type `Bool`

- *Guards* and *conditionals* are alternative styles in programming

Example Algorithm Design — Haskell

Binary Search — Haskell — 2 (b)

```haskell
binarySearchArray :: (Ix i, Integral i, Ord a)
                    => Array i a -> a -> Maybe i
binarySearchArray ary x
  = binarySearchGen p (bounds ary)
    where
    p m = x `compare` (ary ! m)

binarySearchList :: Ord a
                 => [a] -> a -> Maybe Int
binarySearchList xs val
  =  binarySearchGen p (0, length xs - 1)
     where
     p m = val `compare` (xs !! m)
```

Example Algorithm Design — Haskell

Code Description 4

- `compare` is a method of the `Ord` class, for *ordering*, defined in the standard *Prelude*

```
class (Eq a) => Ord a where
  compare              :: a -> a -> Ordering
  (<),(<=),(>=),(>) :: a -> a -> Bool
  max, min             :: a -> a -> a

  compare x y
    | x == y    = EQ
    | x <= y    = LT
    | otherwise = GT

data Ordering = LT | EQ | GT
  deriving (Eq,Ord,Enum,Read,Show,Bounded)
```

- Minimal type-specific definitions required are `compare` or `(==)` and `(<=)`
- `!` and `!!` are the array and list indexing operators

Example Algorithm Design

Binary Search — Haskell — Comparison

- The first version with `binarySearch` and `binarySearch01` is very similar to the *Python* recursive version `binarySearchRec`
- In the *Haskell* case an explicit helper function is used
- The second version is more general: `binarySearchGen` can be used with any type that is indexed by a data type in the `Integral` class
- `binarySearchArray` and `binarySearchList` specialise the function to arrays or lists.
- For the Haskell `Array` data type see the Haskell Report
- Idiomatic Haskell tends to be more general and make use of higher order functions, type classes and advanced features.

Web Links & References

Python IDEs

- Python Online IDEs
  - **Repl.it** https://repl.it/languages/python3 (Read-eval-print loop)
  - **TutorialsPoint CodingGround** Python 3 https://www.tutorialspoint.com/execute_python3_online.php
  - **TutorialsPoint CodingGround** Haskell ghci https://www.tutorialspoint.com/compile_haskell_online.php

Web Links & References

References

- The *offside rule* (using layout to determine the start and end of code blocks) comes originally from Landin (1966) — see Off-side rule for other programming languages that use this.
- The *step-by-step* approach to writing programs is described in Glaser (2000)
- The difficulty in learning programs is described in many articles — see, for example, Dehnadi (2006)
- **Inductive data type**
  - Algebraic data type composite type — possibly recursive sum type of product types — common in modern functional languages.
  - Recursive data type from Type theory