

Computing

Software Install & Summaries

Contents

1	Introduction	1
2	Jupyter Notebook	2
2.1	Notebook Terminology	2
2.2	User Interface: Key Bindings	2
2.2.1	Jupyter Notebook Key Bindings Table Notes	5
3	Python	6
3.1	Python Expressions & Operator Precedence	6
3.1.1	Python Operator Precedence Table Notes	8
3.2	Python Data Types: Complexities	8
4	Haskell	11
4.1	Haskell Operator Precedence & Fixity	11
4.2	Haskell References	11
5	Java	13
5.1	Java Operator Precedence & Associativity	13
5.2	Java Statements	15
5.2.1	Expression & Block Statements	17
5.2.2	Selection Statements	17
5.2.3	Iteration Statements	18
6	JavaScript	20
6.1	JavaScript Operator Precedence & Associativity	20
7	CSS	23
7.1	CSS Links	23
8	macOS	23
8.1	Hidden Files	23
9	Shell Programming	23
9.1	Z Shell	23
9.1.1	Z Shell References	23
10	Web Sites & References	24
	References	24

1 Introduction

These notes give some summaries or *cheatsheets* for various computing courses and related topics — it is not intended for any particular audience other than the author and

includes some topics which may not be directly in any particular course

[ToC](#)

2 Jupyter Notebook

- Jupyter jupyter.org
- Jupyter Notebook Documentation jupyter-notebook.readthedocs.io
- IPython ipython.org
- IPython Documentation ipython.readthedocs.io
- Anaconda anaconda.com
- Anaconda Cloud anaconda.org
- Conda Docs docs.conda.io
- Conda User Guide conda.io/projects/conda/en/latest/user-guide

2.1 Notebook Terminology

- **Notebook Dashboard** When you launch *jupyter notebook* the first page that you encounter is the Notebook Dashboard.
- **Notebook Editor** Once a Notebook is selected to edit, the Notebook will open in the Notebook Editor
- See [User interface components](#)

2.2 User Interface: Key Bindings

- The tables below give *key bindings* for some of the Command Mode and Edit Mode commands
- A more complete set of commands for Command Mode can be seen from [Help](#)
[Edit Keyboard Shortcuts](#)

[ToC](#)

Jupyter Notebook		Key Bindings		macOS				
Modifier & Other Special Keys				macOS				
Cmd		Command		Shift		Space		Up
Ctrl		Control		Caps Lock		Tab		Down
Alt		Option		Return		Del		Right
Esc		Escape		Enter		Back Del		Left

Jupyter Notebook		Command Mode		macOS			
F		Find & replace		Y		Change cell to code	
↵, ⌘		Enter edit mode		M		Change cell to markdown	
⌘ + ⬆ + F		Open command palette		R		Change cell to raw	
⌘ + ⬆ + P		Open command palette		1		Change cell to heading 1	
P		Open command palette		2		Change cell to heading 2	
⬆ + ↵, ⬆ + ⌘		Run cell, select below		3		Change cell to heading 3	
^ + ⌘		Run selected cells		4		Change cell to heading 4	
⇧ + ↵, ⇧ + ⌘		Run cell, insert below		5		Change cell to heading 5	
⬆ + K, ⬆ + ↑		Extend selected cells above		6		Change cell to heading 6	
⬆ + J, ⬆ + ↓		Extend selected cells below		K, ⬆		Select cell above	
⌘ + A		Select all cells		J, ↓		Select cell below	
A		Insert cell above		B		Insert cell below	
X		Cut selected cells		C		Copy selected cells	
⬆ + V		Paste cells above		V		Paste cells below	
Z		Undo cell deletion		D, D		Delete selected cells	
S, ⌘ + S		Save and Checkpoint		⬆ + M		Merge selected cells	
L		Toggle line numbers		⬆ + L		Toggle all cells line numbers	
O		Toggle selected cells output		⬆ + O		Toggle selected cells output scrolling	
I, I		Interrupt the kernel		0, 0		Restart the kernel (with dialog)	
Esc, ⌘, Q		Close the pager		H		Show keyboard shortcuts	
⬆ + ⬇		Scroll notebook up		⬇		Scroll notebook down	

Jupyter Notebook		Edit Mode		macOS	
	→	Code completion or indent	+ + Z	Redo	
+	→	Tooltip	+ + U	Redo selection	
+ J		Indent	+ K	Emacs-style line kill	
+ I		Dedent	+	Delete line left of cursor	
+ A		Select all	+	Delete line right of cursor	
+ Z		Undo	+ M	Enter command mode	
+ /		Comment	esc,	Enter command mode	
+ D		Delete whole line	+ + F	Open command palette	
+ U		Undo selection	+ + P	Open command palette	
Insert (?)	Toggle overwrite flag (?)	+ , +	Run cell, select below		
+ ↑		Go to cell start	+	Run selected cells	
+ ↓		Go to cell end	+ , +	Run cell, insert below	
+ ←		Go one word left	+ + Minus	Split cell at cursor	
+ →		Go one word right	+ S	Save and Checkpoint	
+		Delete word before		Move cursor down	
+		Delete word after		Move cursor up	

Jupyter Notebook	Key Bindings	Windows
Modifier & Other Special Keys		Windows
Menu	Win Menu	Shift
Ctrl	Control	Space
Alt	Alt	Tab
Esc	Escape	Del
	Enter	Back Del
		Up
		Down
		Right
		Left

Jupyter Notebook	Command Mode	Windows
	Find & replace	Change cell to code
	Enter edit mode	Change cell to markdown
+ +	Open command palette	Change cell to raw
+ +	Open command palette	Change cell to heading 1
	Open command palette	Change cell to heading 2
+	Run cell, select below	Change cell to heading 3
+	Run selected cells	Change cell to heading 4
+	Run cell, insert below	Change cell to heading 5
+	Extend selected cells above	Change cell to heading 6
+	Extend selected cells below	, Select cell above
+	Select all cells	, Select cell below
	Insert cell above	Insert cell below
	Cut selected cells	Copy selected cells
+	Paste cells above	Paste cells below
	Undo cell deletion	, Delete selected cells
, +	Save and Checkpoint	+ Merge selected cells
	Toggle line numbers	+ Toggle all cells line numbers
	Toggle selected cells output	+ Toggle selected cells output scrolling
,	Interrupt the kernel	, Restart the kernel (with dialog)
,	Close the pager	Show keyboard shortcuts
+	Scroll notebook up	Scroll notebook down

Jupyter Notebook	Edit Mode	Windows
	Code completion or indent	+ + Redo
+	Tooltip	+ + Redo selection
+	Indent	+ Emacs-style line kill
+	Dedent	+ Delete line left of cursor
+	Select all	+ Delete line right of cursor
+	Undo	+ Enter command mode
+	Comment	, Enter command mode
+	Delete whole line	+ + Open command palette
+	Undo selection	+ + Open command palette
(?) Toggle overwrite flag (?)		+
+	Go to cell start	+ Run selected cells
+	Go to cell end	+
+	Go one word left	+ + Split cell at cursor
+	Go one word right	+ Save and Checkpoint
+	Delete word before	Move cursor down
+	Delete word after	Move cursor up

2.2.1 Jupyter Notebook Key Bindings Table Notes

- The **macOS** colors are:
 - Keys background: yellow **#FFFF00**
 - Odd row background: user defined **myJupyterMacOSBgClrOdd** **#FFFF7F** — this is a tint of yellow — see [color-hex: yellow](#)
 - Even row background: user defined **myJupyterMacOSBgClrEven** **gray!10**
- The **Windows** colors are:
 - Keys background: **PaleTurquoise** **#AFEEEE**
 - Odd row background: user defined **myJupyterWinBgClrOdd** **#C4F2F2** — first monochromatic color — see [color-hex: #AFEEEE](#)
 - Even row background: user defined **myJupyterWinBgClrEven** **gray!10**
- The tables are derived from the *Jupyter Notebook* interface and
 - [Cheatography: Jupyter Notebook Keyboard Shortcuts](#)
 - [Cheatography: Jupyter Notebook Editor Keyboard Shortcuts](#)
 - For both of the above you can download the LaTeX source
- **Queries**
 1. Does Q close the pager (as well as Escape) ?
 2. Does S do a Save and Checkpoint (as well as Ctrl+S) ?
 3. Does Ctrl+Return do Run Selected Cells (as well as Ctrl+Enter) ?
- It may be possible to typeset the tables in Markdown but not done that yet with the colour banding
- **M269 Colors for Marking**
- from 2021J TMA files
- `.answercell{background-color : #FFFFCC;}` **#FFFFCC**
- `.feedbackcell{background-color : #C8ECFF;}` **#C8ECFF**
- `.guidancecell{background-color : #F2C0D4;}` **#F2C0D4**
- See [Converting Colors](#) for display of colors
- FFFFCC is a Websafe version with name conditioner
- Websafe version of C8ECFF is CCFFFF **#CCFFFF**
- Websafe version of F2C0D4 is FFCCCC **#FFCCCC**

3 Python

3.1 Python Expressions & Operator Precedence

The following table is from [Python Reference: Section 6.17 Operator precedence](#) in the Python documentation with highest precedence (most binding) at the top of the table. Operators in the same delimited row are left associative except for exponentiation which is right associative.

If in doubt, keep the brackets — you can still be surprised by some expressions

```
(9 // 2) * (9 // 2) == 16
9 // 2 * 9 // 2    == 18
```

The power operator `**` binds less tightly than an arithmetic or bitwise unary operator on its right (but I would put the brackets in anyway)

```
2**-1 == 2**(-1) == 0.5
```

M269 uses a subset of the operators given below — any student slides/notes would need a cut-down version

The table below include [set operators](#), which are part of the Python Library

[ToC](#)

Python Operator Precedence	
Operator(s)	Description
(<i>expr...</i>), [<i>expr...</i>], { <i>expr...</i> }, { <i>key:value...</i> }	Parenthesized form, Displays for lists, sets, dictionaries (comprehensions) Set displays Dictionary displays
<i>xs[index]</i> , <i>xs[index:index]</i> , <i>f(args...)</i> , <i>o.attr</i>	Sequence subscription, Slicing, Function call, Attribute reference
<i>await expr</i>	Await expression
**	Exponentiation
+ <i>x</i> , − <i>x</i> , ~ <i>x</i>	Unary arithmetic and bitwise operations Unary positive, unary negative, bitwise <i>not</i>
*, @, /, //, %	Binary arithmetic operations Multiplication, matrix multiplication, division, floor division, remainder or <i>string formatting</i>
+, −	Addition, subtraction, <i>set difference</i>
<<, >>	Shifts
&	Binary bitwise operations Bitwise <i>and</i> , <i>set intersection</i>
^	Bitwise <i>xor</i> , <i>set symmetric difference</i>
	Bitwise <i>or</i> , <i>set union</i>
<, <=, >, >=, !=, ==, in, not in, is, is not	Comparisons, <i>set subset</i> , <i>superset</i> , value comparisons, Membership tests, <i>set membership</i> Identity tests
not <i>bexpr</i>	Boolean operations Boolean <i>not</i>
and	Boolean <i>and</i>
or	Boolean <i>or</i>
<i>exprT if bexpr else exprF</i>	Conditional expression
lambda	Lambda expression
:=	Assignment expression

3.1.1 Python Operator Precedence Table Notes

- The table here is derived from [Python Reference: Section 6.17 Operator precedence](#) for version 3.8.3 — the table here is reversed compared to the Python documentation so that the highest precedence is at the top of the table. Versions 3.9 and 3.10 have the table in the order as above
- The odd row background colour is the [HTML color #EEFFCC](#) — the same as the code background colour in the Python documentation
- [Lutz \(2013, Table 5-2, page 137\)](#) also has `yield expr` at the lowest precedence — parentheses probably required everywhere except *parentheses may be omitted when the yield expression is the sole expression on the right hand side of an assignment statement*.

[ToC](#)

3.2 Python Data Types: Complexities

- The following tables are based on wiki.python.org/moin/TimeComplexity
- **Health Warning** these notes are work in progress and need to be read with the Python documentation
- [Sequence Type — list, tuple, range](#)
- [Set Types — set, frozenset](#)
- [Mapping Types — dict](#)
- [collections.deque](#)
- The complexities here may have some errors or typos or be misleading — **beware**

[ToC](#)

Python Data Types: Complexities			List
Operation		Average	Amortized Worst
Get item	<code>x = xs[i]</code>	$O(1)$	$O(1)$
Set item	<code>xs[i] = x</code>	$O(1)$	$O(1)$
Append	<code>xs = ys + zs</code>	$O(1)$	$O(1)$
Copy	<code>xs = ys[:]</code>	$O(n)$	$O(n)$
Pop last	<code>xs.pop()</code>	$O(1)$	$O(1)$
Pop other	<code>xs.pop(i)</code>	$O(k)$	$O(k)$
Insert(i,x)	<code>xs[i:i] = [x]</code>	$O(n)$	$O(n)$
Delete item	<code>del xs[i:i+1]</code>	$O(n)$	$O(n)$
Get slice	<code>xs = ys[i:j]</code>	$O(k)$	$O(k)$
Set slice	<code>xs[i:j] = ys</code>	$O(k + n)$	$O(k + n)$
Delete slice	<code>xs[i:j] = []</code>	$O(n)$	$O(n)$
Member	<code>x in xs</code>	$O(n)$	
Get length	<code>n = len(xs)</code>	$O(1)$	$O(1)$
Count(x)	<code>n = xs.count(x)</code>	$O(n)$	$O(n)$
Sort	<code>ys = sorted(xs)</code>	$O(n \log n)$	$O(n \log n)$
min(xs), max(xs)	<code>y = min(xs)</code>	$O(n)$	

Complexities: Set		Python		Haskell
Operation		Average	Worst	Worst
Member	<code>x in s</code>	$O(1)$	$O(n)$	$O(\log n)$
Union	<code>s t</code>	$O(m + n)$		$O(m \log(\frac{n}{m} + 1))$
Intersection	<code>s & t</code>	$O(\min(m, n))$	$O(m \times n)$	$O(m \log(\frac{n}{m} + 1))$
Difference	<code>s - t</code>	$O(m)$		$O(m \log(\frac{n}{m} + 1))$
Insert	<code>s.add(x)</code>	$O(1)$	$O(n)$	$O(\log n)$
Delete	<code>s.discard(x)</code>	$O(1)$	$O(n)$	$O(\log n)$
Remove	<code>s.remove(x)</code>	$O(1)$	$O(n)$	$O(\log n)$

Python Data Types: Complexities			Dict
Operation		Average	Amortized Worst
Member	<code>k in d</code>	$O(1)$	$O(n)$
Get item	<code>d.get(k)</code>	$O(1)$	$O(n)$
Set item	<code>d.get(k)</code>	$O(1)$	$O(n)$
Delete	<code>d.pop(k, default)</code>	$O(1)$	$O(n)$
Copy	<code>d.copy()</code>	$O(n)$	$O(n)$
Iterate	<code>iter(d)</code>	$O(n)$	$O(n)$

Python Data Types: Complexities		collections.deque	
Operation		Average	Amortized Worst
Copy	<code>dq.copy()</code>	$O(n)$	$O(n)$
Append	<code>dq.append(x)</code>	$O(1)$	$O(1)$
Append left	<code>dq.appendleft(x)</code>	$O(1)$	$O(1)$
Pop	<code>dq.pop()</code>	$O(1)$	$O(1)$
Pop left	<code>dq.popleft()</code>	$O(1)$	$O(1)$
Extend	<code>dq.extend(iterable)</code>	$O(k)$	$O(k)$
Extend left	<code>dq.extendleft(iterable)</code>	$O(k)$	$O(k)$
Rotate	<code>dq.rotate(n=1)</code>	$O(k)$	$O(k)$
Remove	<code>dq.pop(k, default)</code>	$O(n)$	$O(n)$

4 Haskell

4.1 Haskell Operator Precedence & Fixity

- In the following table
- **P** Precedence — 9 highest, 0 lowest, default 9
- **A** Fixity, associativity — **L** Left, **N** Non-associative, **R** Right
- The table is based on the [Haskell 2010 Language Report](#) (Section 4.4.2 Fixity Declarations) — note that the Haskell libraries have more operators than are in the table

4.2 Haskell References

- [FP Complete Haskell Tutorial: Operator Glossary](#)
- [Hackage Haskell Cheat Sheet](#)

[ToC](#)

Haskell Operator Precedence				
Operator(s)		Type	Description	A P
!!	[a] -> Int -> a		List index	L 9
.	(b -> c) -> (a -> b) -> a -> c		Function composition	R 9
!	Ix i => Array i e -> i -> e		Array index (Data.Array)	L 9
//	Ix i => Array i e -> [(i, e)] -> Array i e		Array update (Data.Array)	L 9
^	(Integral b, Num a) => a -> b -> a		Exponentiation	R 8
^^	(Fractional a, Integral b) => a -> b -> a		Exponentiation	R 8
**	Floating a => a -> a -> a		Exponentiation	R 8
*	Num a => a -> a -> a		Multiply	L 7
/	Fractional a => a -> a -> a		Divide	L 7
'div', 'mod', 'quot', 'rem'	Integral a => a -> a -> a		Integer division	L 7
%	Integral a => a -> a -> Ratio a		Ratio (Data.Ratio)	L 7
+, -	Num a => a -> a -> a		Addition, Subtraction	L 6
<>	Semigroup a => a -> a -> a		Associative op (Semigroup)	L 6
:	a -> [a] -> [a]		List constructor	R 5
++	[a] -> [a] -> [a]		List append	R 5
\\	Eq a => [a] -> [a] -> [a]		List difference (Data.List)	N 5
==, /=	Eq a => a -> a -> Bool		Equality	N 4
<, <=, >, >=	Ord a => a -> a -> Bool		Comparison	N 4
'elem', 'notElem'	(Foldable t, Eq a) => a -> t a -> Bool		Membership	N 4
<\$>	Functor f => (a -> b) -> f a -> f b		Infix fmap (Data.Functor)	L 4
<*>	Applicative f => f (a -> b) -> f a -> f b		Sequential application (Control.Applicative)	L 4
&&	Bool -> Bool -> Bool		Logical <i>and</i>	R 3
	Bool -> Bool -> Bool		Logical <i>or</i>	R 2
>>	Monad m => m a -> m b -> m b		Compose two actions	L 1
>>=	Monad m => m a -> (a -> m b) -> m b		Compose two actions	L 1
\$\$, \$!	(a -> b) -> a -> b		Application (strict)	R 0
'seq'	a -> b -> b		Strict eval	R 0

5 Java

5.1 Java Operator Precedence & Associativity

- In the table of expressions and operator precedence and associativity, operators in the same group of rows have higher precedence than those below
- The column **A** represents **operator associativity**
 - **L** left associative
 $e1 + e2 + e3$ means $(e1 + e2) + e3$
 - **R** right associative
 $e1 = e2 = e3$ means $e1 = (e2 = e3)$
 - **N** not associative
 $1 < x < 4$ is not valid
- *integer* means [char](#), [byte](#), [short](#), [int](#), [long](#), or the boxed forms Character, Byte, Short, Integer, Long
- *numeric* means integer or [float](#), [double](#) or the boxed forms Float, Double
- The table is based on [Sestoft \(2016, section 11.1, page 37\)](#)
- [Evans and Flanagan \(2018, page 35\)](#) or [Evans et al. \(2023, page 36\)](#) gives a similar table
- The information would be derived from [Java Language Specification: Chapter 15. Expressions](#) but that is not an easy read

Java Operator Precedence					
Expression	Meaning	A	P	Arg Types	Result Types
<code>a[...]</code>	array access		16	<code>t[]</code> , integer	<code>t</code>
<code>o.f</code>	non-static field access			object <code>o</code>	type of <code>f</code>
<code>C.f</code>	static field access				type of <code>f</code>
<code>this</code>	current object reference				this class
<code>o.m(...)</code>	instance method call			object <code>o</code>	return type
<code>C.m(...)</code>	static method call				return type
<code>super.m(...)</code>	superclass method call				return type
<code>C.super.m(...)</code>	encl. superclass meth. call				return type
<code>t.class</code>	class object for <code>t</code>			type <code>t</code>	<code>Class<t></code>
<code>t::m</code> or <code>e::m</code>	method reference				
<code>x++</code> or <code>x--</code>	postincrement/decrement			numeric	numeric
<code>++x</code> or <code>--x</code>	preincrement/decrement		15	numeric	numeric
<code>-x</code>	negation	R		numeric	numeric
<code>~e</code>	bitwise complement	R		integer	<code>int/long</code>
<code>!e</code>	logical negation	R		boolean	boolean
<code>new t[...]</code>	array creation		14	type <code>t</code>	<code>t[]</code>
<code>new C(...)</code>	object creation			class <code>C</code>	<code>C</code>
<code>(t) e</code>	type cast			type, any	<code>t</code>
<code>*, /</code>	multiplication, division	L	13	numeric	numeric
<code>e1 % e2</code>	remainder	L		numeric	numeric
<code>+, -</code>	addition, subtraction	L	12	numeric	numeric
<code>e1 + e2</code>	string concatenation	L		String, any	String
<code>e1 + e2</code>	string concatenation	L		any, String	String
<code>e1 << e2</code>	left shift	L	11	integer	<code>int/long</code>
<code>e1 >> e2</code>	signed right shift	L		integer	<code>int/long</code>
<code>e1 >>> e2</code>	unsigned right shift	L		integer	<code>int/long</code>
<code><, <=, >=, ></code>	comparison	N	10	numeric	boolean
<code>e instanceof t</code>	instance test	N		any, reference type	boolean
<code>e1 == e2</code>	equal	L	9	compatible	boolean
<code>e1 != e2</code>	not equal	L		compatible	boolean
<code>e1 & e2</code>	bitwise and	L	8	integer	<code>int/long</code>
<code>e1 & e2</code>	logical strict and	L		boolean	boolean
<code>e1 ^ e2</code>	bitwise exclusive-or	L	7	integer	<code>int/long</code>
<code>e1 ^ e2</code>	logical strict exclusive-or	L		boolean	boolean
<code>e1 e2</code>	bitwise or	L	6	integer	<code>int/long</code>
<code>e1 e2</code>	logical strict or	L		boolean	boolean
<code>e1 && e2</code>	logical and	L	5	boolean	boolean
<code>e1 e2</code>	logical or	L	4	boolean	boolean
<code>e1 ? e2 : e3</code>	conditional	L	3	boolean, any, any	any
<code>x = e</code>	assignment	R	2	<code>e</code> subtype of <code>x</code>	type of <code>x</code>
<code>x += e</code>	compound assignment	R		compatible	type of <code>x</code>
<code>x -> ebs</code>	lambda expression	R	1		

5.2 Java Statements

- A *statement* is the basic unit of execution in Java
- A statement may change the computer's *state*: the value of variables, fields and array elements; the contents of files; and so on
- The execution of a statement can:
 - terminate normally (meaning execution will continue with the next statement, if any)
 - terminate abruptly by throwing an exception
 - exit by executing a `return` statement (if inside a method or constructor)
 - exit a switch or loop by executing a `break` (if inside a switch or loop)
 - exit the current iteration of a loop and start a new iteration by executing a `continue` statement or
 - does not terminate at all (eg, `while (true) {}`)
- The table of Java statements below is based on (Evans and Flanagan, 2018, page 51) and (Evans et al., 2023, page 52)

Java Statements		
Statement	Purpose	Syntax
expression	side effects	<code>var = expr;expr++;method();new Type();</code>
compound	group stmnts	<code>{ stmnts }</code>
empty	do nothing	<code>;</code>
labelled	name stmnt	<code>label : stmnt</code>
if	conditional	<code>if (expr) stmnt [else stmnt]</code>
switch	conditional	<code>switch (expr) {case expr : stmnts ...default: stmnts}</code>
switch	conditional expr	15.28 Switch Expressions
while	loop	<code>while (expr) stmnt</code>
do	loop	<code>do stmnt while (expr);</code>
for	simplified loop	<code>for (init; test; increment) stmnt</code>
foreach	collection iteration	<code>for (var : iterable) stmnt</code>
break	exit block	<code>break [label]</code>
continue	restart loop	<code>continue [label]</code>
return	end method	<code>return [expr]</code>
synchronized	critical collection	<code>synchronized (expr) {stmnts}</code>
throw	throw exception	<code>throw expr ;</code>
try	handle exception	14.20 The try statement
try	h.e. closing resources	14.20.3 try-with-resources
assert	verify invariant	<code>assert [: error] ;</code>
yield	return specified value	14.21 The yield Statement

5.2.1 Expression & Block Statements

- An expression statement is an **expression** followed by a **;**

```
expression ;
```

- The only forms of **expression** that may be used here are assignments, increment and decrements, method call, and object creation
- A block statement is a sequence of variable declarations, class declarations and statements

```
{  
  variableDeclarations  
  classDeclarations  
  statements  
}
```

- An empty statement consists of **;** only — it is equivalent to the block statement **{ }**

[ToC](#)

5.2.2 Selection Statements

- The **if** statement has the form

```
if (condition)  
  trueBranch
```

- The **if-else** statement has the form

```
if (condition)  
  trueBranch  
else  
  falseBranch
```

- The **condition** must have type **boolean** or **Boolean**
- **trueBranch** and **falseBranch** are statements
- What is wrong with the following

```
if (dataAvailable) ;  
  processData() ;
```

```
if (dataAvailable)  
  processData() ;  
  reportResults() ;
```

```
if (dataAvailable)  
  processData() ;  
  reportResults() ;  
else  
  reportNoData() ;
```

```
if (dataAvailable) ;  
  processData() ;
```

- The **trueBranch** is an empty statement (**;**)

```
if (dataAvailable)  
  processData() ;  
  reportResults() ;
```

- `reportResults()` ; will always be executed

```
if (dataAvailable)
    processData() ;
    reportResults() ;
else
    reportNoData() ;
```

- Will not compile
- **Moral** Always use block statements
- A `switch` statement has the form

```
switch (expression) {
    case constant1: branch1
    case constant2: branch2
    ...
    default: branchN
}
```

- `expression` must be of type `int`, `short`, `char`, `byte` or a boxed version of these or `String` or an enum type
- Each `constant` must be a compile-time constant expression, consisting only of literals, `final` variables, `final` fields declared with explicit field initialisers or an unqualified enum value
- (not used in M250)

[ToC](#)

5.2.3 Iteration Statements

- A `for` statement has the form

```
for (initialization ; condition ; step)
    body
```

- `initialization` is a `variableDeclaration` or an expression
- `condition` is an expression of type `boolean` or `Boolean`
- `step` is an expression
- `body` is a statement
- `initialization` and `step` may be comma-separated lists of expressions
- `initialization`, `condition` and `step` may be empty. An empty `condition` is equivalent to `true`
- The `for` statement is executed as follows
 1. The `initialization` is executed
 2. The `condition` is evaluated. If it is `false`, the loop terminates.
 3. If it is `true` then
 - (a) the `body` is executed
 - (b) the `step` is executed
 - (c) execution continues at (2.)

- What does the following code do ?

```
for (int i = 1 ; i <= 4 ; i++) {
    for (int j = 1 ; j <= i ; j++) {
        System.out.print("*") ;
    }
    System.out.println() ;
}
```

```
jshell> for (int i = 1 ; i <= 4 ; i++) {
...>     for (int j = 1 ; j <= i ; j++) {
...>         System.out.print("*") ;
...>     }
...>     System.out.println() ;
...> }
...>
*
**
***
****
```

- A **while** statement has the form

```
while (condition)
    body
```

- **condition** is an expression of type **boolean** or **Boolean** and **body** is a statement
- It is executed as follows:

1. The **condition** is evaluated. If it is **false**, the loop terminates
2. If it is **true**, then
 - (a) The **body** is executed
 - (b) Execution continues at (1.)

- Example linear search with **while** loop

```
String[] wdays =
{"Monday", "Tuesday", "Wednesday",
 "Thursday", "Friday", "Saturday", "Sunday"} ;

int wdayno(String wday) {
    int i = 0 ;
    while (i < wdays.length
        && ! wday.equals(wdays[i])) {
        i++ ;
    }
    if (i < wdays.length) {
        return i ;
    } else {
        return -1 ;
    }
}
```

```
String[] wdays =
{"Monday", "Tuesday", "Wednesday",
 "Thursday", "Friday", "Saturday", "Sunday"} ;

int wdayno(String wday) {
    int i = 0 ;
    while (i < wdays.length
        && ! wday.equals(wdays[i])) {
        i++ ;
    }
    if (i < wdays.length) {
        return i ;
    } else {
        return -1 ;
    }
}
```

```
jshe11> int d1 = wdayno("Friday") ;
d1 ==> 4

jshe11> int d2 = wdayno("Dimanche") ;
d2 ==> -1
```

- Write code using a **while** statement that is equivalent to a **for** loop statement

```
initialization
while (condition) {
    body
    step
}
```

```
for (initialization ; condition ; step)
    body
```

- Note that this is different behaviour to the **for** statement in Python where assignments to variables in the suite of the loop does not change the assignments made in the target list
- See [Python: for statement](#)
- A **foreach** statement has the form

```
for (tx x : expression)
    body
```

- The **expression** must have type `Iterable<t>` where **t** is a subtype of **tx**

[ToC](#)

6 JavaScript

6.1 JavaScript Operator Precedence & Associativity

- [Mozilla Developers Network: JavaScript](#)
- [Mozilla Developers Network: JavaScript reference](#)
- The following table is based on [Mozilla Developer Network: JavaScript Reference: Operator precedence](#)
- The column **A** represents **operator associativity**
 - **R** right associative

$e1 = e2 = e3$ means $e1 = (e2 = e3)$
 - **L** left associative

$e1 + e2 + e3$ means $(e1 + e2) + e3$

$3 > 2 > 1$ is valid and evaluates to **false** — why?
 - **N** not associative
- For information about APIs see
 - [Web APIs](#)
 - [Document Object Model \(DOM\)](#)



JavaScript Operator Precedence			
Operator(s)	Description	A	P
<i>(expr...)</i>	Grouping		21
<i>o.prop</i>	Member Access	L	20
<i>o[propStr]</i>	Computed Member Access	L	
<i>new C(args)</i>	<i>new</i> (with arg list)		
<i>f(args)</i>	Function Call	L	
<i>?.</i>	Optional chaining	L	
<i>new C</i>	<i>new</i> (without arg list)	R	19
<i>x++,x--</i>	post-increment/decrement		18
<i>! b</i>	Logical NOT	R	17
<i>~ x</i>	Bitwise NOT	R	
<i>+x, -x</i>	Unary plus, negation	R	
<i>++x, --x</i>	pre-increment/decrement	R	
	<i>typeof,void,delete,await</i>	R	
<i>x ** y</i>	Exponentiation	R	16
<i>*,/,%</i>	Multiplication,Division,Remainder	L	15
<i>+, -</i>	Addition,Subtraction	L	14
<i><<,>>,>>></i>	Bitwise Left Shift,Right,Unsigned	L	13
<i><,<=,>,>=</i>	Comparisons	L	12
	<i>in,instanceof</i>	L	
<i>==,!=,===,!==</i>	Equality, Strict Equality	L	11
<i>&</i>	Bitwise AND	L	10
<i>^</i>	Bitwise XOR	L	9
<i> </i>	Bitwise OR	L	8
<i>&&</i>	Logical AND	L	7
<i> </i>	Logical OR	L	6
<i>??</i>	Nullish coalescing operator	L	5
<i>b ? x : y</i>	Conditional	R	4
<i>=</i>	Assignment	R	3
<i>op=</i>	Assignment with operation	R	
	<i>yield,yield*</i>	R	2
<i>expr1 , expr2</i>	Comma/Sequence	L	1

7 CSS

7.1 CSS Links

- [CSS Snapshot 2018](#)
- [Selectors Level 4 W3C Working Draft 21 November 2018](#)

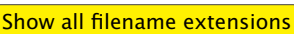
[ToC](#)

8 macOS

8.1 Hidden Files

- Some files and folders in macOS are not displayed by default in [Finder](#) or [Terminal](#)
- These are files with names starting with a dot ([.](#)), [Library](#) folders (there are several) and some system folders.
- Here are several ways of making these files visible
- **Finder (1)** with [Finder](#) selected, type  +  +  — the keystroke command is a toggle so to turn viewing off just re-type the same
- **Finder (2)** to make the change permanent, type the following in [Terminal](#)

```
defaults write com.apple.Finder AppleShowAllFiles true  
killall Finder
```

- **Finder (3)** to make a permanent change without using [Terminal](#) have a look at [TinkerTool](#) from <https://www.bresink.com/osx/TinkerTool.html> — this is free and does lots of other tweaks — the [Finder](#) tab first item is [Show hidden and system files](#)
- Also make sure you display filename extensions with    and check 

[ToC](#)

9 Shell Programming

9.1 Z Shell

9.1.1 Z Shell References

- [Z shall](#)
- [Welcome to Zsh](#) redirects to above
- [Oh My Zsh](#)
- [Z shell macOS setup](#)
- [Bash vs Z shell: A Tale of Two Command Line Shells](#)
- [Moving to zsh](#)
 - [Part 1: Moving to zsh](#)

- [Part 2: Configuration Files](#)
- [Part 3: Shell Options](#)
- [Part 4: Aliases and Functions](#)
- [Part 5: Completions](#)
- [Part 6: Customizing the zsh Prompt](#)
- [Part 7: Miscellanea](#)
- [Part 8: Scripting zsh](#)
- **zsh first use wizard** [Run the Zsh first use wizard](#) (6 September 2022)
- [Startup and termination files used by the various Unix shells](#)
- [Why does macOS Catalina use Zsh instead of Bash? Licensing](#)
- [Conda User Guide: conda init](#)
- [Anaconda: Installing on macOS](#)

[ToC](#)

10 Web Sites & References

References

- Bloch, Joshua (2017). *Effective Java*. Addison-Wesley Professional, third edition. ISBN 9780134685991.
- Evans, Benjamin J.; Jason Clark; and David Flanagan (2023). *Java in a Nutshell: A Desktop Quick Reference*. O'Reilly Media, eighth edition. ISBN 1098131002. [13](#), [15](#)
- Evans, Benjamin J and David Flanagan (2018). *Java In A Nutshell*. O'Reilly, seventh edition. ISBN 1492037257. [13](#), [15](#)
- Lutz, Mark (2013). *Learning Python*. O'Reilly, fifth edition. ISBN 1449355730. URL <http://learning-python.com/books/about-1p5e.html>. [8](#)
- Martelli, Alex; Anna Martelli Ravenscroft; Steve Holden; and Paul McGuire (2022). *Python in a Nutshell: A Desktop Quick Reference*. O'Reilly, fourth edition. ISBN 1098113551.
- Ramalho, Luciano (2022). *Fluent Python: Clear, Concise, and Effective Programming*. O'Reilly. ISBN 1492056359.
- Sestoft, Peter (2016). *Java Precisely*. MIT, third edition. ISBN 0262529076. URL <http://www.itu.dk/people/sestoft/javaprecisely/>. [13](#)

[ToC](#)