Lecture 2: Variables and Expressions

COMP90059 Introduction to Programming

Wally Smith

School of Computing & Information Systems

Lecture Overview

- In Week 1, we jumped straight in to look at **variables** and **variable assignment** in Python, and we started to experiment with the functions **print** and **input**
- In Week 2, we will keep looking at how to use these techniques, but now we need to understand more about the following things:
  - understanding the different elements that make up code
  - the naming of variables
  - the importance of data types
  - string expressions
  - arithmetic expressions
  - the Days of Life calculation - developing a program in parts
- Workshops start this week - Week 2
- Reference. John Zelle (2017) Python Programming 3rd Ed. Franklin Beedle.

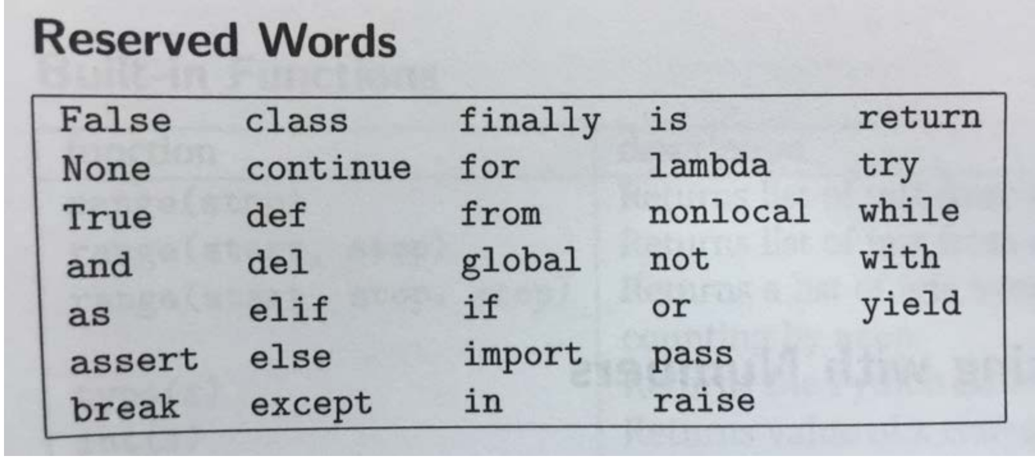
Variables and their names

The different elements of code ...

```
principal = int(input('Enter the initial principal: '))
apr = float(input('Enter the annual interest rate: '))
numberYears = int(input('Enter the number of years: '))
apr = apr/100
for i in range(numberYears):
        principal = principal * (1 + apr)
print('The final value is: ', round(principal))
```

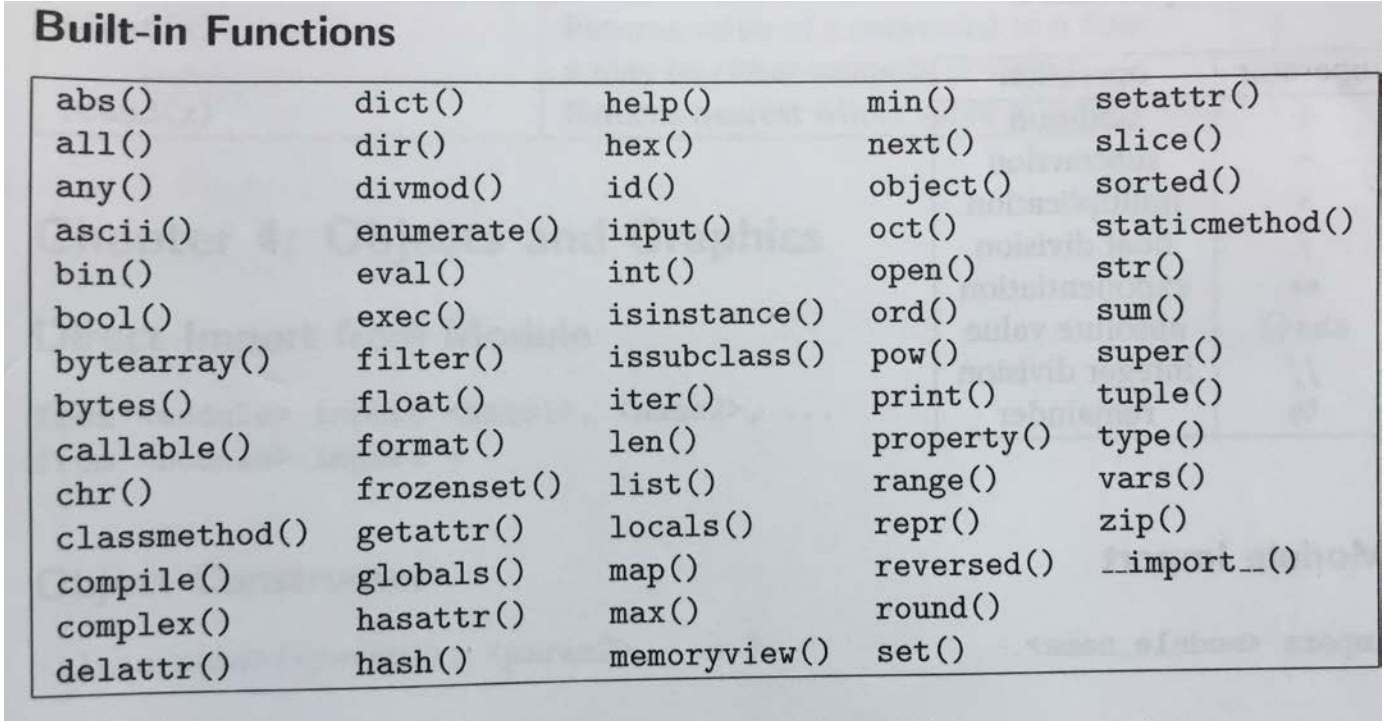
built-in functions, of Python
reserved words, of Python
variable names, created by programmer
prompt, created by programmer
literals, created by programmer

Reserved Words & Built-in Functions



Reserved Words

False
None
True
and
as
assert
break
class
continue
def
del
elif
else
except
finally
for
from
global
if
import
in
is
lambda
nonlocal
not
or
pass
raise
return
try
while
with
yield



Built-in Functions

abs()
all()
any()
ascii()
bin()
bool()
bytearray()
bytes()
callable()
chr()
classmethod()
compile()
complex()
delattr()
dict()
dir()
divmod()
enumerate()
eval()
exec()
filter()
float()
format()
frozenset()
getattr()
globals()
hasattr()
hash()
help()
hex()
id()
input()
int()
isinstance()
issubclass()
iter()
len()
list()
locals()
map()
max()
memoryview()
min()
next()
object()
oct()
open()
ord()
pow()
print()
property()
range()
repr()
reversed()
round()
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()
tuple()
type()
vars()
zip()
__import__()

Zelle, 2017

Variable Names - Rules in Python

- Reserved words cannot be used as variable names

  Examples: if, for, while and import

- Don't use built-in functions as variable names

  Examples: enumerate, print, input

- Names must begin with a letter or underscore _

- The rest of the name can contain zero or more occurrences of the following things:

  digits (0 to 9)
  alphabetic letters
  underscores

- Names are case sensitive

  Example: **WEIGHT** is different from **weight**

Exercise 1. Variable Name Quiz

- #Which of the following are valid variable names?

a. length
b. continue
c. x
d. _width
e. firstWord
f. first_word
g. 2MoreToGo
h. halt!
i. JOURNEY

Variable Naming Conventions

When choosing names for variables ...

- be succinct
  - find the most meaningful word or term that most clearly and unambiguously identifies what the variable represents in the task
  - but also try to keep it short!
  - depends on what other variables you need to define
    - amount - might be a good name in one program, but confusing in another
- use lower case with “camel casing” for multiple words
  - Example: target, interestRate, daysFirstYear
- use all uppercase letters for symbolic constants
  - Examples: TAX_RATE and STANDARD_DEDUCTION

Exercise 2: Refresher of L01

- Write a program that asks the user where they live (taking this into a variable) and then uses the variable to print a message to say that wherever they live is a wonderful place.

Exercise 3: Refresher of L01

- Write a program that asks the user to enter any noun, any verb and any adjective, and then prints to the screen a sentence using this data.

Data types

Data Types

Type of data	Python type name	Examples (literals)
Integers	int	-1, 0, 6895
Real numbers	float	-0.101011, 567738.009187
Character strings	str	””, ’d’, ”I’m a string”, ‘3’
Boolean	bool	True, False

- 'The data type of an item determines what values it can hold and what operations it supports' (Zelle, 2017)

- A literal is a specific or 'actual' data value

String (str)

- Strings consists of a sequence of characters.

- Python represents strings in data type: str

- Strings include all the alphanumeric letters and numbers, plus special characters - depending on what characters set is supported.
- Examples: 'xI99Pjz_fff2356', 'Hello', '435', 'FishPond45'

- Data provided by the user is interpreted first by Python as a string because it is entered as a sequence of characters typed into the keyboard.

Integers (int)

- Integers: zero, all positive whole numbers, and all negative whole numbers
- Python represents these in data type: int
- A leading negative sign indicates a negative value in python

- Examples :   34, 2, -45677, 0, 409000
- Integer literals (like the examples above) in Python are written without commas

Limits ...

- A computer's memory places a limit on the magnitude of the largest positive and negative integers
- Python's int typical range: -2^{31} to $2^{31} - 1$ ie., (-2147483648 to 2147483647)
- Try evaluating: 2147483647 ** 100

Floating-point numbers (float)

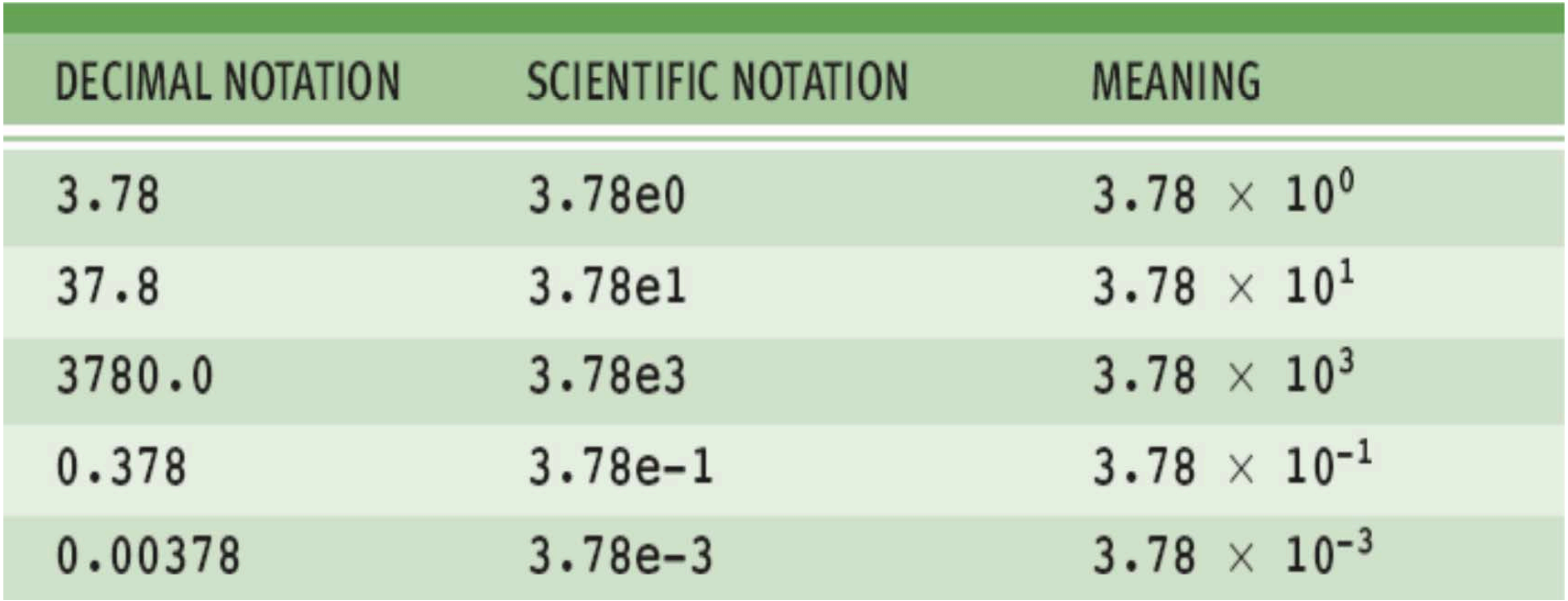
- A real number consists of a whole number, a decimal point and fractional part.
- Python uses floating-point numbers to represent real numbers - data type: float
- A leading negative sign indicates a negative value in python

- Examples: 1.45, 455.0405, 1.00, -98.5401
- A floating point number can be written using either ordinary decimal notation or scientific notation (see next slide)
- Scientific notation is useful when representing very large numbers

Limits ...

- Python’s float typical range: $–10^{308}$ to 10^{308}

Floating-point numbers



DECIMAL NOTATION	SCIENTIFIC NOTATION	MEANING
3.78	3.78e0	3.78×10^{0}
37.8	3.78e1	3.78×10^{1}
3780.0	3.78e3	3.78×10^{3}
0.378	3.78e-1	3.78×10^{-1}
0.00378	3.78e-3	3.78×10^{-3}

Literals

- **Literals** are specific or actual pieces of raw data in the program code:
  - **'Cindy'** (a string literal)
  - **45** (an integer literal)
  - **11. 34** (a floating-point literal)

Exercise 4 : Data Types

- **Which data type would most appropriately be used to represent the following data values?**

a. The number of people who visit a company's website

b. The average time spent on the website by each visitor

c. The area of a circle

d. The approximate age of the universe

e. A password

f. A company profit

g. A reason for why a decision was made

h. A football player's team number

- **Write the values of the following floating point numbers in Python’s scientific notation:**

a. 77.89

b. 0.0000529

Dynamic typing

- A distinctive feature of Python is that it decides what types a variable is when you first assign a value to it ...

```
name = 'Cindy'
```

Python creates a variable called name, and designates it as a string variable, and associates the variable with the literal value 'Cindy'

Type conversion

- To “cast” a literal or variable to a different type, we use functions of the same name as the type:
- `int(), float(), str()`

```
amountAUDstr = input('Enter amount in Australian Dollars and Cents: ')
amountAUD = float(amountAUDstr)
rateAUDtoIC = 84.32
amountIK = 84.32 * amountAUD
print('Equivalent to',amountIK,'in Icelandic Krona')
```

```
age = int( input('Enter your name: ') )
```

Exercise 5. What is the output of the following statements?

a) print(int(34.56))

b) print(int(1.75))

c) print(int(-1.75))

d) print(int('3.45'))

e) print(str(34.56))

f) print(str(10))

g) print(float(4))

h) print(float('3.45'))

i) print(float('abc'))

String Expressions

Expressions

Expressions are 'fragments of code that produce or calculate new data values' (Zelle, 2017)

- name * 5
- yearOfBirth + yearsElapsed
- timeAtRaceEnd - timeAtRaceStart

Some String expressions

- concatenation

```
print(‘a’ + ‘b’ + 'c' )
abc
```

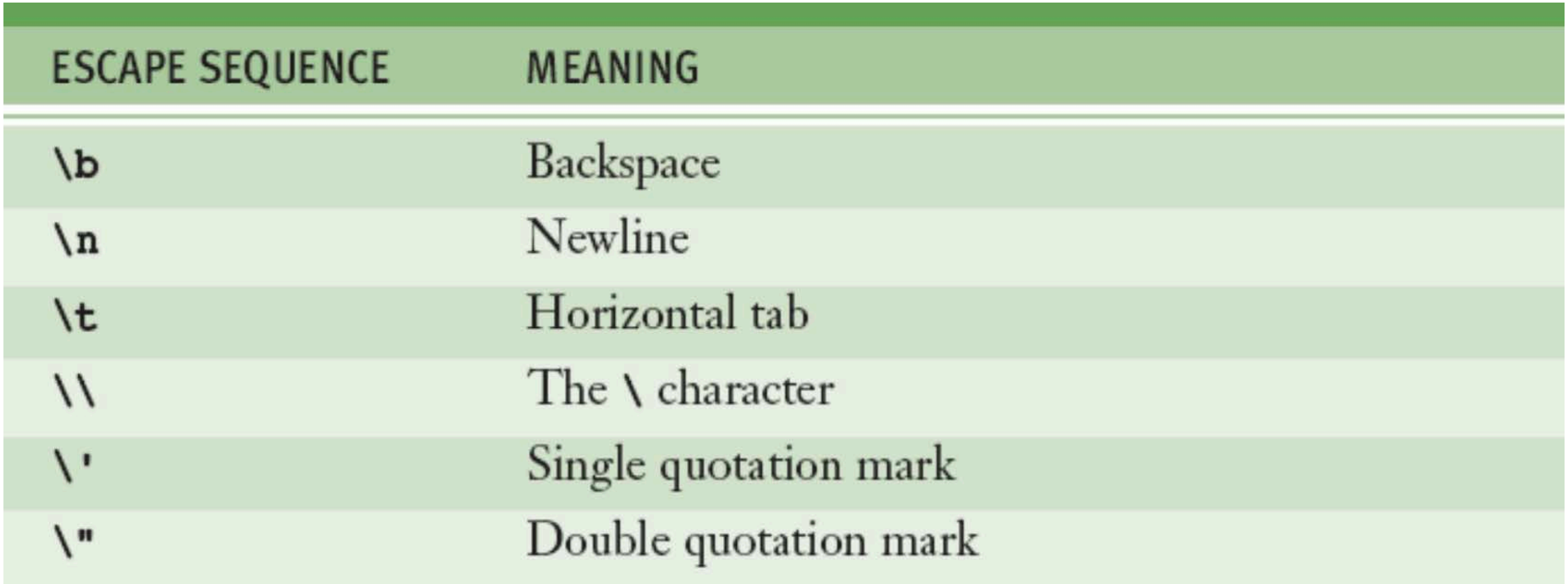
- repetition

```
print('z' * 20)
zzzzzzzzzzzzzzzzzzzz
```

- in (subset)

```
print('z' in 'zizzer zazzer zuzz')
True
```

Escape Sequences



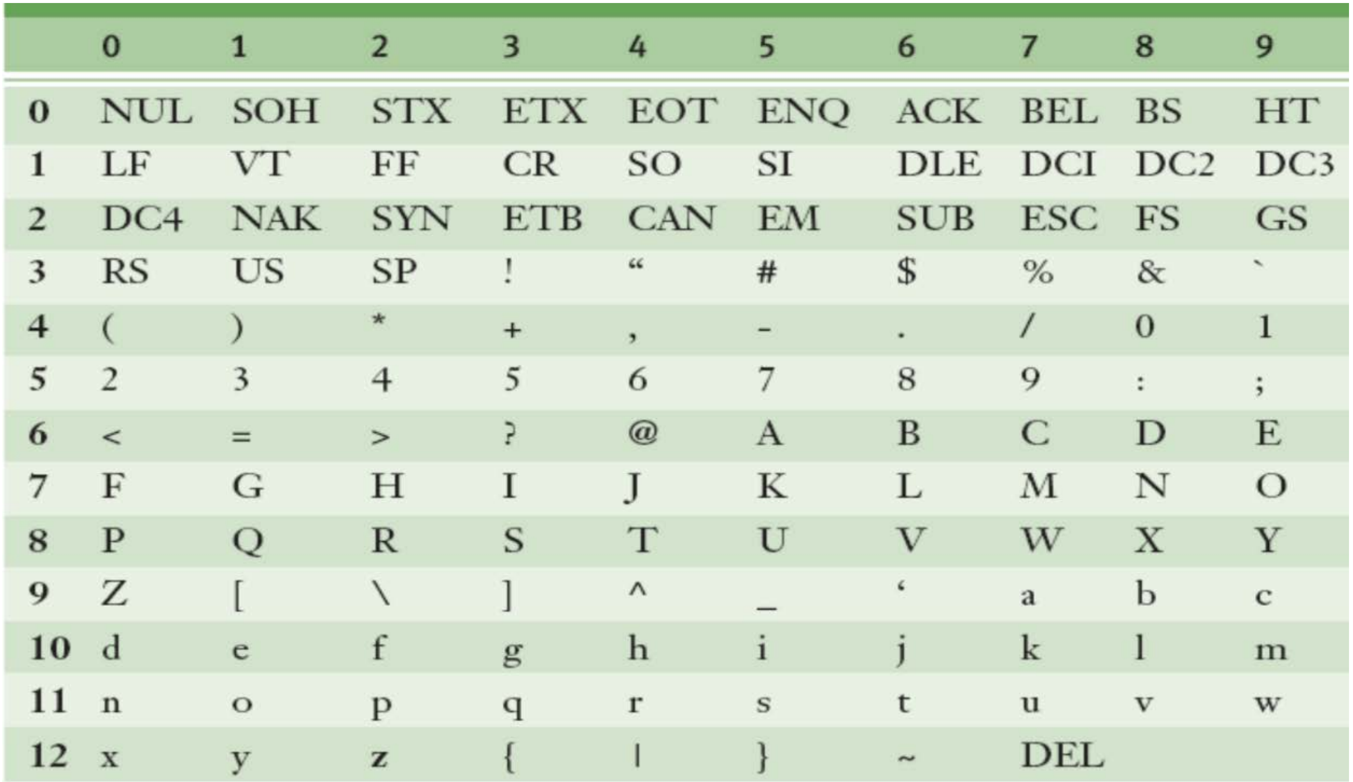
ESCAPE SEQUENCE	MEANING
`\b`	Backspace
`\n`	Newline
`\t`	Horizontal tab
`\\`	The \ character
`\'`	Single quotation mark
`\"`	Double quotation mark

```
print('Happy Birthday to you\nHappy Birthday to you')
```

```
Happy Birthday to you
Happy Birthday to you
```

Character Sets, & the chr and ord functions

- In Python, character literals look just like string literals and are of the string type
  - They belong to several different character sets, among them the ASCII set (ASCII character set maps to set of integers)
- ord and chr convert characters to and from ASCII



	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	“	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

```
>>> ord(‘a’)
     97

>>> chr(100)
     ‘d’
```

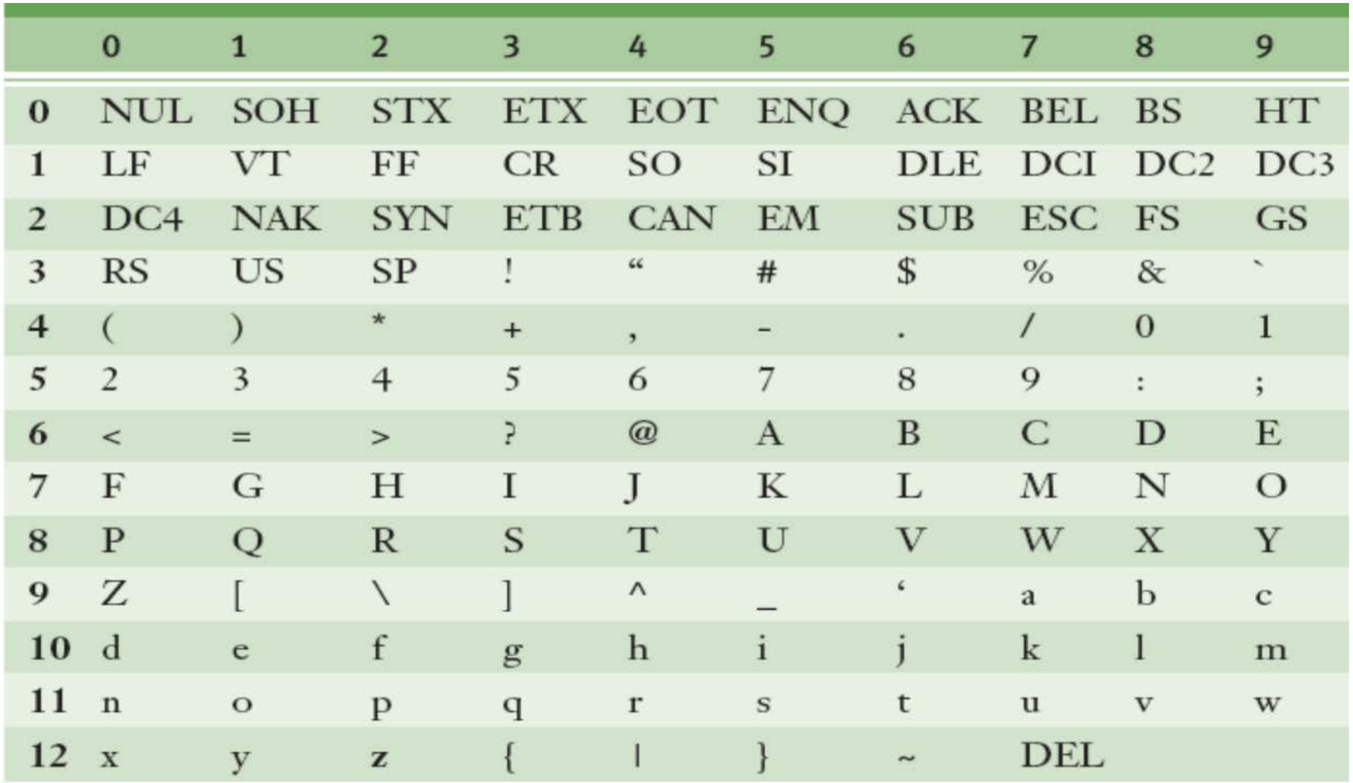
Exercise 6 : Character conversions

- **What is the output of the following?**

a. print(chr(108))

b. print(ord(‘H’))

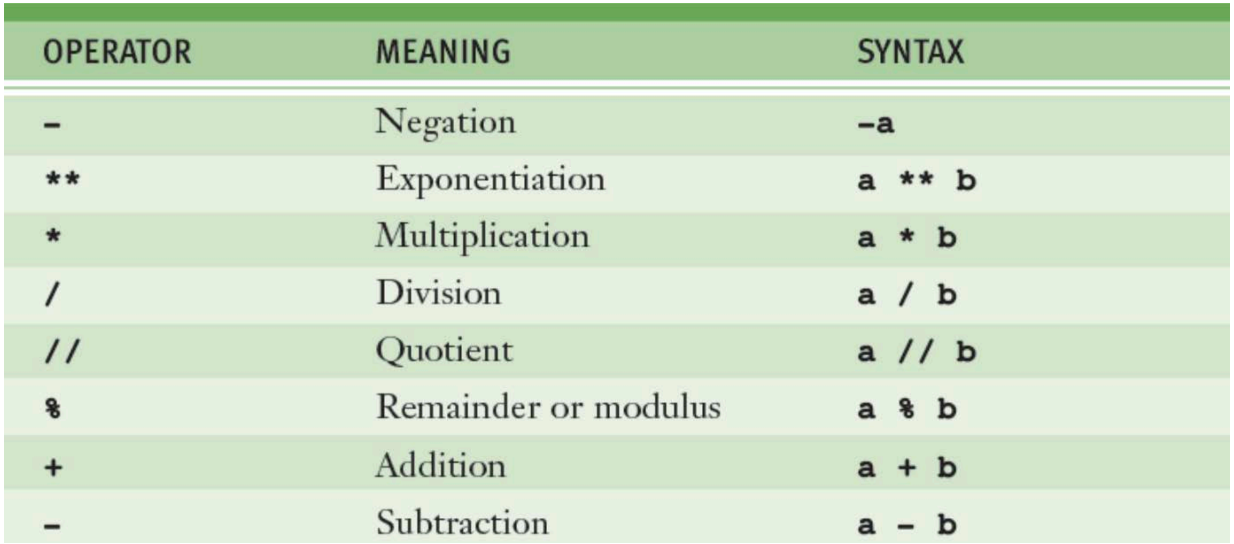
c. print(chr(ord(‘Q’) + 4))



	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	“	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Numeric Expressions

Arithmetic Expressions



OPERATOR	MEANING	SYNTAX
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

The area of a square of sides 7 cms:

```
area = 7 ** 2
```

The number of 6-packs of eggs that can be completely filled by 33 eggs:

```
numberBoxes = 33 // 6
```

The number of eggs left over

```
numberExtraEggs = 33 % 6
```

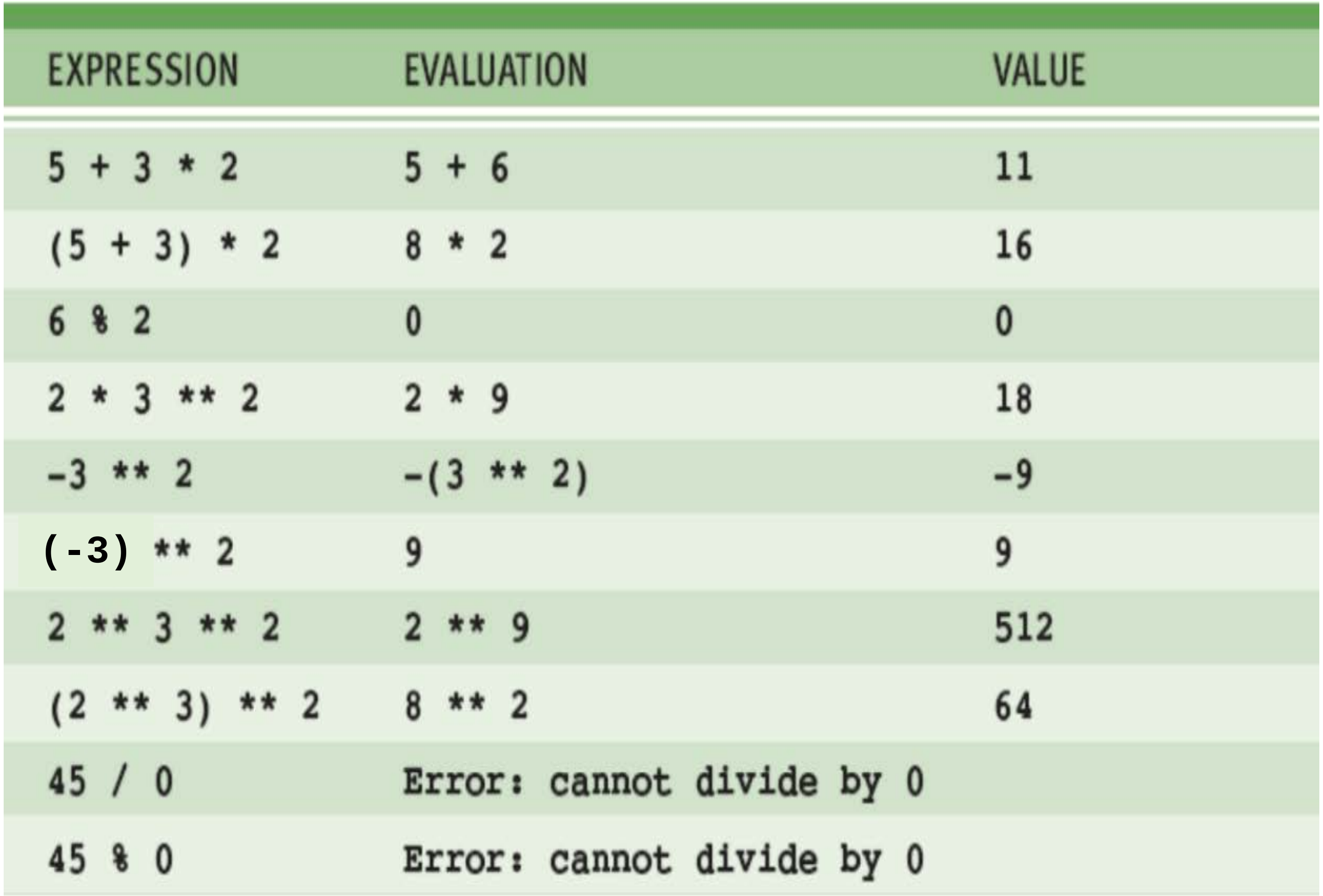
Arithmetic expressions

Precedence rules (in this order)

parentheses (first)	( )
exponentiation	**
unary negation	–
multiplication, division, remainder	*, / , //, %
addition, subtraction (last)	+ , –

- operations of equal precedence are **left associative**, evaluated from left to right
- exponentiation (**) and assignment (=) are **right associative**

Evaluating arithmetic expressions



EXPRESSION	EVALUATION	VALUE
5 + 3 * 2	5 + 6	11
(5 + 3) * 2	8 * 2	16
6 % 2	0	0
2 * 3 ** 2	2 * 9	18
-3 ** 2	-(3 ** 2)	-9
(-3) ** 2	9	9
2 ** 3 ** 2	2 ** 9	512
(2 ** 3) ** 2	8 ** 2	64
45 / 0	Error: cannot divide by 0	
45 % 0	Error: cannot divide by 0	

right associative

Exercise 7. Evaluating arithmetic expressions

#Let x = 7 and y = 3. Write the values of the following expressions:

a. x + y * 3
b. (x + y) * 3
c. x ** y ** 2
d. x % y
e. (x - 1) / 12.0
f. x // 6

Exercise 8 :   What is the output?

```
print(10/3)

print (10.0/3.0)

print(10 / 5)
```

```
print(10//3)

print(10.0//3.0)

print(10%3)

print(10.0 % 3.0)
```

(Zelle, 2017)

Exercise 9 :   Write a program to ...

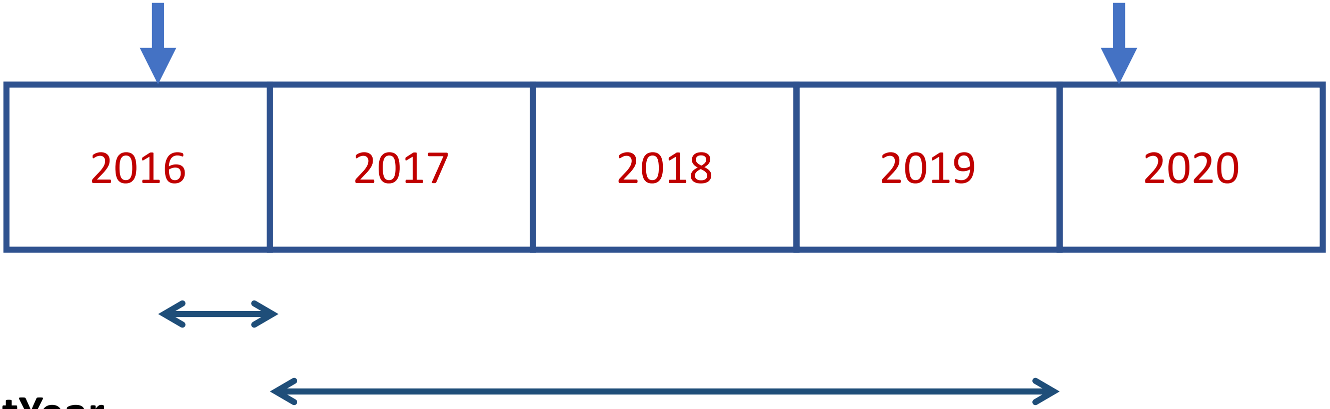
# Take a person's birth date and then tell them approximately how many days they have been alive.

Analyzing the days of life calculation

Take someone who was born on 8 September 2016 ...

birth date
08 September 2016

today
09 March 2020



daysFirstYear
* 30 - 8 days of Sept left
* 12 - 9 = 3 months left
@ 30 days each

daysWholeYears
* years = 2020 - 2016 - 1
* @365 days per year

daysCurrentYear
* whole months = 3 - 1 @
30days each
* 9 days of March

```
# Take a person's birth date and then tell them approximately how many days they have been alive.
```

Exercise 10

Look through the code for the Days of Life calculation.
Write down every example of the following elements.

built-in functions, of Python
variable names, created by programmer
prompt, created by programmer
reserved words, of Python
literals, determined by programmer

Reminders from Lecture 1

Install Python

- We will use Python v3.6 or later
- Programs are developed in Python IDLE (Interactive DeveLopment Environment)
- You just enter your code as text and the Python interpreter turns it into machine code for you
- Get a copy of python for your own machine at home - there are free versions for Windows, MacOS and Linux
  - http://www.python.org/download/
- Portable version (USB) http://portablepython.com/
- Advanced Python Distribution (for scientific experimentation)
  - http://www.enthought.com/products/edudownload.php

Grok Learning environment

- GROK Learning is the web-based programming environment we will be using for the duration of this subject in your labs:

  https://groklearning.com/course/unimelb-comp90059-2020-s1/
- All you need to access the system is a browser, an internet connection and your GROK account
- Different modes of working in GROK: code, run, mark, terminal
- To access GROK, you will need to login using your university email