COMPSCI 105 S1 2017
Principles of Computer Science

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

Introduction

Learning outcomes

- A student who successfully completes this course will be able to:
  - Define a class to model and represent an object
  - Write code which handles important exception types
  - Use a standard data interchange format for reading and writing complex data types
  - Write programs that store and manipulate data in standard linear data structures (arrays, linked lists, stacks, queues) and non-linear data structures (hash tables, trees)
  - Compare the efficiency of algorithms using standard big-O notation
  - Implement recursive solutions to simple problems
  - Implement recursive data structures such as linked lists and trees
  - Explain the basic algorithm for any of the studied sorting methods
  - use regular expressions to extract data from a body of text

Lecturers & Tutors

- Lecturers
  - Angela Chang (Course coordinator)
    - Email: angela@cs.auckland.ac.nz
    - Phone: 3737599 ext 86620
    - Room: 303.414
    - Office hours: whenever the office door is open
  - Dr Bruce Sham
    - Email: b.sham@auckland.ac.nz
    - Phone: 3737599 ext 87387
    - Room: 303S-588
    - Office hours: TBA
  - Dr Burkhard Wuensche
    - Email: burkhard@cs.auckland.ac.nz
    - Phone: 3737599 ext 83705
    - Room: 303-529
    - Office hours: TBA

- Tutors
  - Lindsay Shaw
    - Email: lsha074@aucklanduni.ac.nz
  - Teererai Marange
    - Email: t.marange@auckland.ac.nz

Assessment

- **Note: Students must obtain a pass in both the practical (assignments) and non-practical work (test + exam) in order to pass as a whole**
- Practical ………………………………………….25%
  - 10 Laboratories (1% each)
  - 3 Assignments (5% each)
- Mid-semester Test ……………………………………15%
  - Monday 3rd April, 6:15pm-7:15pm
  - Email Angela (angela@cs.auckland.ac.nz) if you are unable to attend the test.
  - The test is 60 minutes long plus 5 minutes of reading time.
- Final Exam ……………………………………….60%
  - Date to be announced

Laboratories

- All Laboratories will be started from Monday 13 Mar.
  - You must attend an hour tutorial lab sessions each week. You should attend the same lab times each week.
  - There are 10 labs and each lab is worth 1% of your final mark.
  - Venue: B75
    - Thursday 5:00pm-6:00pm
    - Friday 9:00am-10:00am
    - Friday10:00am-11:00am
    - Friday11:00am-12:00noon
    - Friday 5:00pm-6:00pm
  - At your lab time you will be given programming problems to solve.

Code Runner

- The CodeRunner tool is designed to help you practise by presenting you with a set of **coding** and other exercises. Students can work with online exercises using the Moodle learning system.
- Information about using CodeRunner is available on CompSci 105 assignments web page
  - https://www.coderunner.auckland.ac.nz/moodle/

Assignments

- Assignments
  - There are 3 assignments in total worth 15% of your final mark.
  - You are required to write and submit one or more programs.
  - Assignments are handed in using the Assignment Drop Box
    - https://adb.auckland.ac.nz/Home

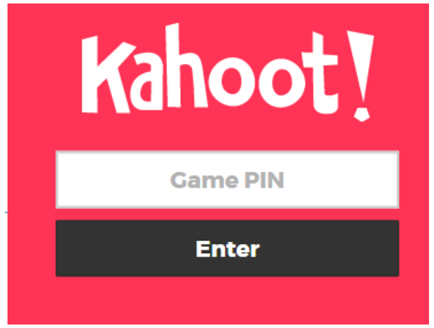
Resources

- Lecture slides
  - https://www.cs.auckland.ac.nz/courses/compsci105s1c/lectures/
- Lecture Recordings
  - Note: All **marks**, **lecture recordings** and **announcements** can be found on the Canvas system. https://canvas.auckland.ac.nz
- Forum
  - Question and answers – peers, tutors and lecturers
  - https://forums.cs.auckland.ac.nz/
- Textbook
  - Problem Solving with Algorithms and Data Structures using Python
  - Online, free, open source
    - http://interactivepython.org/runestone/static/pythonds/index.html
- Additional resources
  - Python.org
  - PythonTutor.com
  - https://www.cs.auckland.ac.nz/courses/compsci105s1c/resources/
    - For information about resources, textbook, references, assessment, people involved in the course and lots more

Class Representative

- Must elect a class rep
- Attends 2 staff student meetings
- Pass on student concerns to lecturers

Kahoot



- Create, play and share fun learning games for …
- How to play
  - On their personal devices, players can then join by going to kahoot.it in their web browser(on install the kahoot app to your own device) , and entering the pin displayed on the screen at the front of the room
  - They then enter their nickname, seeing it displayed at the front
  - They then use their device to answer each question, with the aim to get as many points as possible and get to the top of the leaderboard
  - https://www.youtube.com/watch?v=v2JbY979WUg
- Let's start the first one…

Revision – Python Programs

- Python is a programming language designed to be easy to read
  - Each step in the program is known as a statement
  - A program is a sequence of statements
- Ways of running a program
  - Interactive execution – great for learning
  - Creating a module (file) and executing the module
- Download from http://python.org/download/
- Python comes with a large library of standard modules
- There are several options for an IDE
  - IDLE – works well with Windows
  - Emacs with python-mode or your favorite text editor
  - Eclipse with Pydev (http://pydev.sourceforge.net/)
  - Notepad++

Variables

- Variables store information
  - Information is divided into different types
  - Python is dynamically typed
  - Variables do not need to be declared before they are used

```
x = 34
x = 34.5
x = True
x = 'Hello'
```

- Basic types
  - Integers
  - Floats
  - Strings
    - Can use "" or " to specify with "abc" == 'abc'
    - Use triple double-quotes for multi-line strings or strings than contain both ' and " inside of them:

```
x = 3.456
```

```
z = 5 / 2
```

```
2.5
```

```
z = 5 // 2
```

```
2
```

```
"""a'b"c"""
```

```
a'b"c
```

Assignment

- Binding a variable in Python means setting a name to hold a reference to some object
  - Assignment creates references, not copies
- Names in Python do not have an intrinsic type, objects have types
  - Python determines the type of the reference automatically based on what data is assigned to it
- You can assign to multiple names at the same time
  - This makes it easy to swap values
  - Assignments can be chained

```
x, y = 2, 3
a = b = x = 2
```

Tracing code

- Keep track of the contents of variables
  - Write down the name of each variable
  - Change the value when (and only when) an assignment occurs
  - When you change a value, cross out the old one and write a new one

length_in_inches: ~~50~~ 100

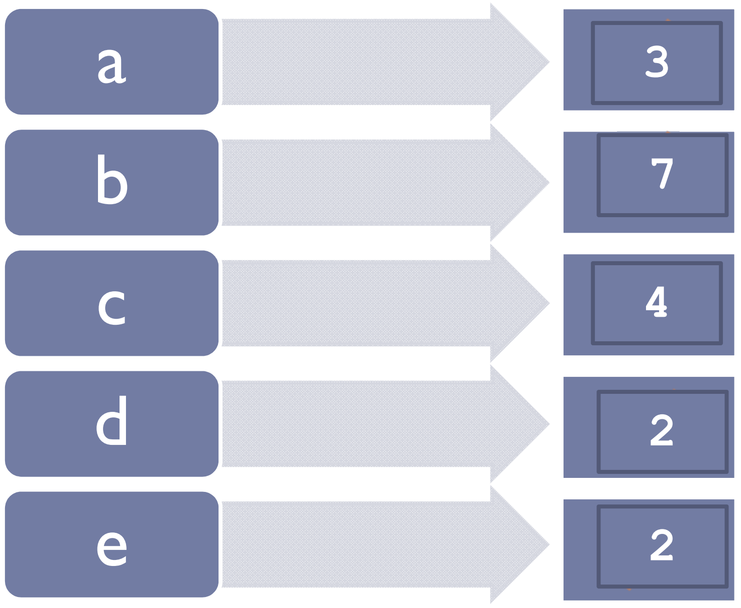
length_in_cms: 254.0

Example 1

Example01.py

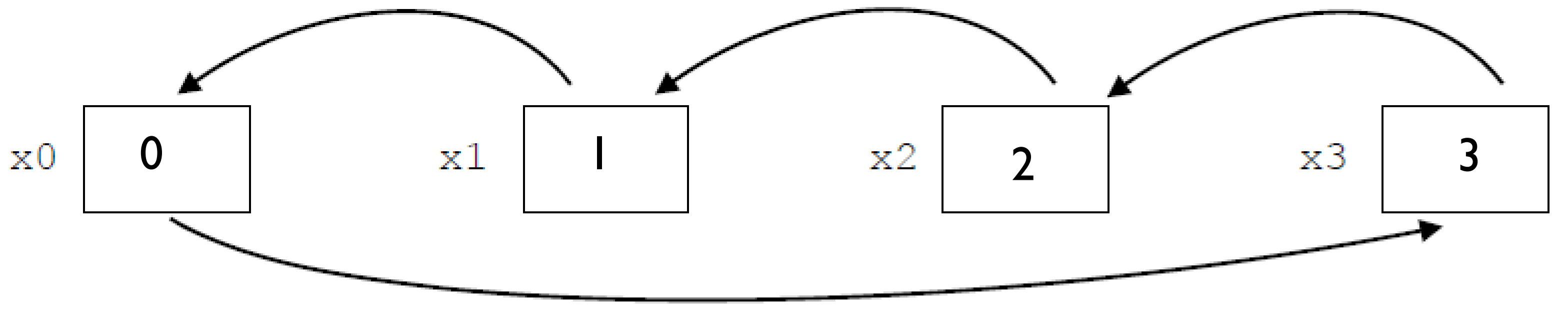
- What is the output of the following code? Perform a code trace.

```
a = 7
b = 3
c = 2
d = 4
e = a

a = b
b = e
e = c
c = d
d = e
print(a, b, c, d, e)
```



Exercise 1

- Suppose that there are 4 variables names x0, x1, x2 and x3. Write the code to move the values stored in those variables to the left, with the leftmost value ending up in the rightmost variable, as shown in the diagram below.



```
print (x0, x1, x2, x3)
```

```
1 2 3 0
```

Accessing Non-Existent Name

- Accessing a name before it's been properly created (by placing it on the left side of an assignment), raises an error

```
>>> y
Traceback (most recent call last):
  ...
NameError: name ‘y' is not defined

>>> y = 3
>>> y
3
```

```
Example01.py
```

The print function *

- The print statement has been replaced with a print() function
  - Elements separated by commas print with a space between them

```
Old: print "The answer is", 2*2
New: print("The answer is", 2*2)
```

- You can also customize the separator between item

```
print("There are <", 2**32, "> possibilities!")
print("There are <", 2**32, "> possibilities!", sep="")
```

```
…are < 4294967296 > poss…
…are <4294967296> poss…
```

- By default, a newline ("\n") is written after the last value in args. You may specify a different line terminator, or no terminator at all.

```
print(3, end=' ')
print(4, end=' ')
print('hello')
```

```
3 4 hello
```

Exercise 2

- Which of the following will not produce "helloworld" in the output?

```
print("hello", end="")
print("world")
```

```
print("hello", "world")
```

```
print("hello", "world", sep="")
```

```
print("helloworld")
```

Expression

- An expression is part of the program that can be evaluated (i.e. when the computer follows the rules that define the instructions, an expression turns into a value).

```
x = 3 + 4
```

3 + 4 is an expression

- An expression can be used anywhere that a value is used

Example

Example01.py

- Floor division and modulus
  - Integer part of a division, and remainder after division

- What do each of the following expressions evaluate to?

10 + 4	14
10 - 4	6
10 * 4	40
10 / 4	2.5
10 ** 4	10000
10 // 4	2
10 % 4	2

Boolean values and related operators

- Boolean values
  - True
  - False
- Relational operators
  - >, >=, <, <=, ==
- Boolean operators
  - and, or, not

```
>>> 2 == 3
```

```
False
```

```
>>> 2 == 3 or 4 < 5
```

```
True
```

Conditionals

`Example01.py`

- Code is executed if the condition is true

name	**Andrew**
n	6

```
if name == "Andrew":
    print("Hi Andrew")
```

```
Hi Andrew
```

```
if n % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```

```
Even number
```

```
if n < 0:
    print("Negative number")
elif n > 0:
    print("Positive number")
else:
    print("Zero")
```

```
Positive number
```

Functions

- A function is a sequence of instructions designed to perform a task, and is packaged as a unit.
  - Functions have a name
  - Functions accept arguments/parameters
  - Functions return values

```
def rectangle_area(width, height):
    return width * height
```

- Syntax
  - Indentation rather than braces are used to signify blocks of code
  - Variables defined within the scope of a function are not available outside the function

Exercise 3

- Write a function that calculates the area of a circle
  - area = πr^2

`Example01.py`

Arguments: Default values

- Parameters can be assigned with default values
  - If the function is called without the argument, the argument gets its default value.
  - They are overridden if a parameter is given for them.
  - The type of the default doesn't limit the type of a parameter.

```
def foo(x=3):
   print(x)
```

```
foo()
```

```
foo(10)
```

```
foo('hello')
```

```
3
10
hello
```

`Example01.py`

Arguments: Named

- Arguments can be specified in any order by using named arguments.
  - Note: any positional arguments must be come before named ones in a call.
  - Example:

```
def info(value, spacing=10, collapse=1):
```

Two optional arguments

```
info(5)
info(5, 12)
info(5, collapse=0)
info(spacing=15, value=5)
```

```
5 10 1
5 12 1
5 10 0
5 15 1
```

```
info(spacing=15, 5)
```

SyntaxError: non-keyword arg after keyword arg

Input

- The input(string) function returns a line of user input as a string
  - The parameter is used as a prompt
  - The string can be converted by using the conversion methods int(string), float(string), etc.

```
x = int(input("Enter an integer: "))
y = int(input("Enter another integer: "))
sum = x+y
print(sum)
```

```
Enter an integer: 3
Enter another integer: 2
5
```

File Input & Output

inflobj = open('data', 'r')	Open the file 'data' for input
S = inflobj.read()	Read whole file into one String
S = inflobj.read(N)	Reads N bytes (N >= 1)
L = inflobj.readlines()	Returns a list of line strings

outflobj = open('data', 'w')	Open the file 'data' for writing
outflobj.write(S)	Writes the string S to file
outflobj.writelines(L)	Writes each of the strings in list L to file
outflobj.close()	Closes the file

Python Operator Precedence

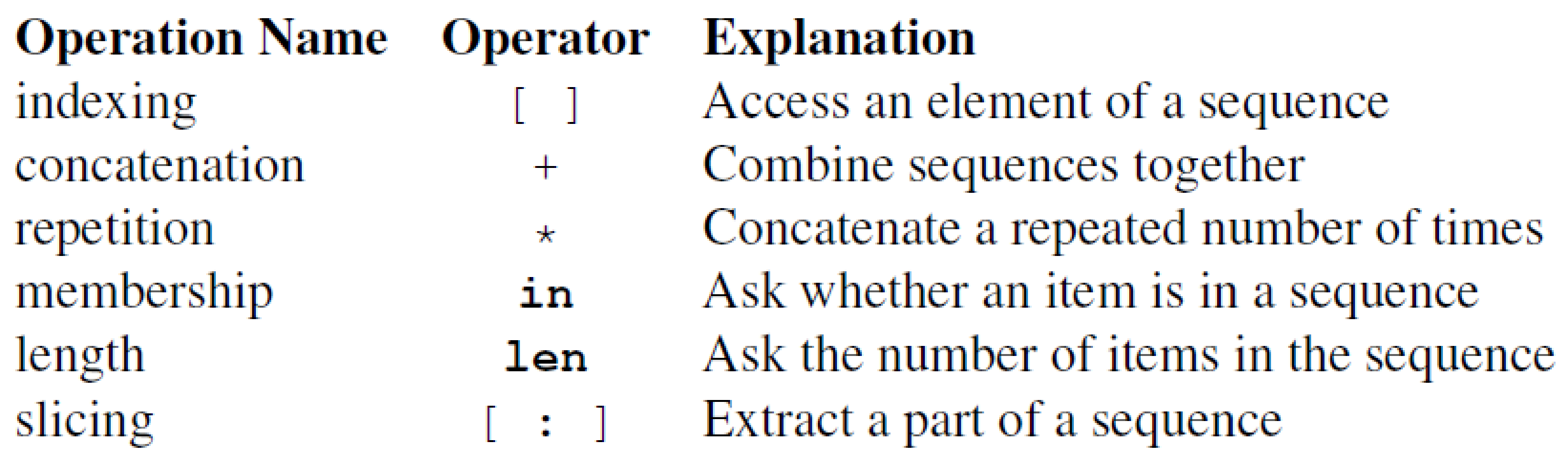
Operator	Description
`()`	Parentheses (grouping)
`f(args...)`	Function call
`x[index:index]`	Slicing
`x[index]`	Subscription
`x.attribute`	Attribute reference
`**`	Exponentiation
`~x`	Bitwise not
`+x, -x`	Positive, negative
`*, /, %`	Multiplication, division, remainder
`+, -`	Addition, subtraction
`<<, >>`	Bitwise shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`in, not in, is, is not, <, <=, >, >=, <>, !=, ==`	Comparisons, membership, identity
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`lambda`	Lambda expression

Sequences

- Sequences allow you to store values in an organized fashion.
  - Tuple: (‘john’, 32, [CMSC])
    - A simple **immutable** ordered sequence of items
    - Items can be of mixed types, including collection types
  - Strings: “John Smith”
    - Immutable
  - List: [1, 2, ‘john’, (‘up’, ‘down’)]
    - **Mutable** ordered sequence of items of mixed types

Similarity and Difference

- All three sequence types (tuples, strings, and lists) share much of the same syntax and functionality.



Operation Name	Operator	Explanation
indexing	[ ]	Access an element of a sequence
concatenation	+	Combine sequences together
repetition	*	Concatenate a repeated number of times
membership	in	Ask whether an item is in a sequence
length	len	Ask the number of items in the sequence
slicing	[:]	Extract a part of a sequence

- Key difference:
  - Tuples and strings are immutable
  - Lists are mutable

Strings

- Strings are a sequence of characters

name | Andrew

```
>>> name = 'Andrew'
>>> name[0]
```

```
'A'
```

```
>>> 'd' in name
```

```
True
```

```
>>> len(name)
```

```
6
```

```
>>> name + ' ' + 'Luxton-Reilly'
```

```
'Andrew Luxton-Reilly'
```

```
>>> 'hello' * 3
```

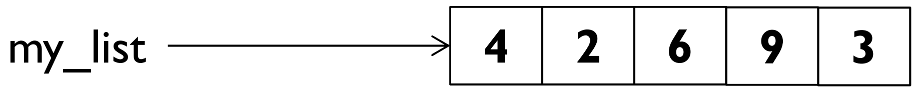
```
hellohellohello
```

- Strings also have a number of other functions that can be used
  - split() is especially useful

Lists

- Lists are a built-in type in Python
  - Use square brackets to signify a list
  - Lists can contain any type of data, or any mixture of data

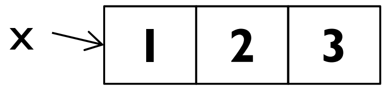
```
my_list1 = [1, 2, 3]

my_list2 = ['Hello', 'Is', 'there', 'anybody', 'out', 'there?']

my_list3 = [1, 5.899, 'Hello']

my_list4 = [4, 2, 6, 9, 3]
```



List functions

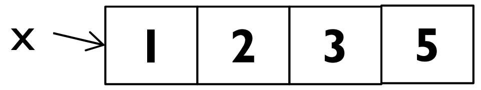
- Numerous list functions are supported
  - Use help(list) to find out the functions
  - Examples:

```
>>> x = [1, 2, 3]
>>> len(x)
3
```



```
>>> x + [4]
[1, 2, 3, 4]
```

```
>>> x += [5]
[1, 2, 3, 5]
```



```
>>> 3 in x
True
```

```
>>> x[0]
1
```

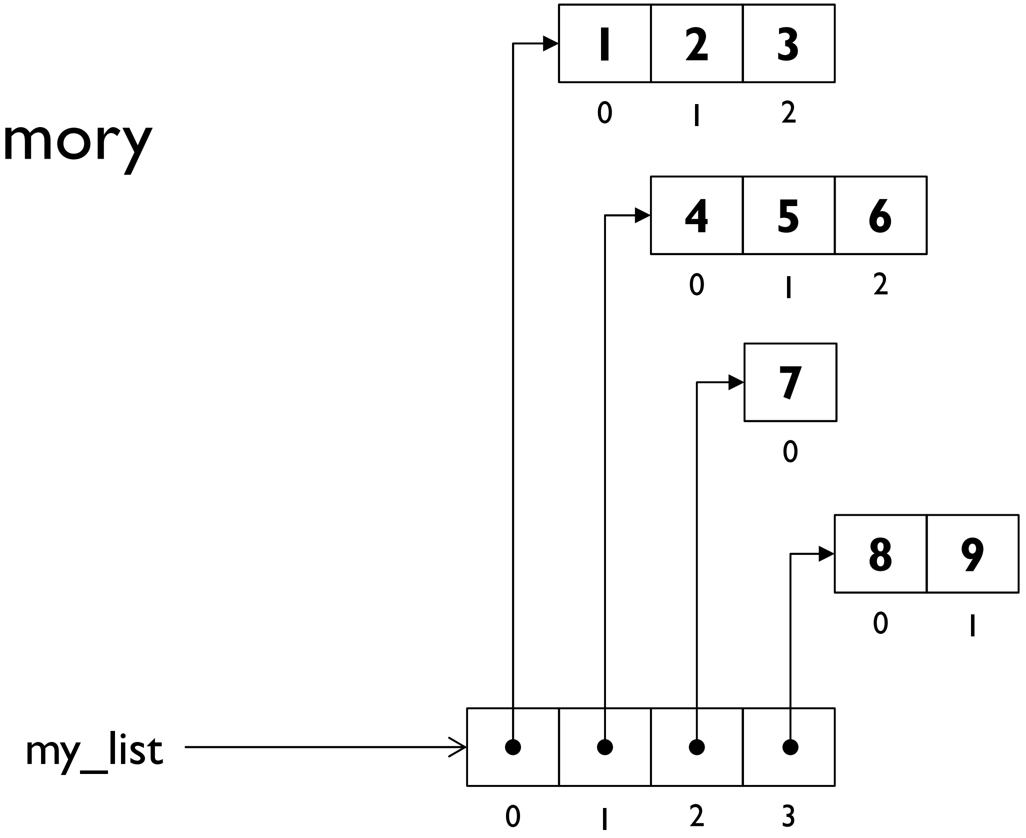
```
>>> [1, 2, 3] * 2
[1, 2, 3, 1, 2, 3]
```

Lists of lists

- Since a list can contain anything, it can of course contain a list

```
my_list = [[1, 2, 3], [4, 5, 6], [7], [8, 9]]
```

- In memory



Tuples

- Tuples are **immutable**
- Define tuples using parentheses and commas
- In order to make a tuple with one element:
  - ',' is needed to differentiate from the mathematical expression (2)
- Examples:

```
>>> tu = (23, 'abc', 4.5)
>>> len(tu)
3
```

```
>>> tu + (2,)
(23, 'abc', 4.5, 2)
```

```
>>> 23 in tu
True
```

```
>>> tu[0]
23
```

```
>>> tu * 2
(23, 'abc', 4.5, 23, 'abc', 4.5)
```

Slices of sequences

- A piece of a sequence can be obtained using the following syntax
  - sequence_name[x:y]
  - where x is the index of the first element and y is the index after the last element

```
>>> name = 'Andrew'
>>> name[0:0]
```

''

```
>>> name[0:1]
```

'A'

```
>>> name[1:4]
```

'ndr'

Slice step value

- Actually, the syntax allows for a third value, used to define the step size between elements included in the slice. If a value if omitted, it defaults to [start:end:1]

```
>>> name = 'Andrew'
>>> name[:4:2]
```

```
'ad'
```

name	A	n	d	r	e	w
Positive Index	0	1	2	3	4	5
Negative index	-6	-5	-4	-3	-2	-1

```
>>> name = 'Andrew'
>>> name[::-1]
```

```
'werdnA'
```

- If the step size is **negative**, it starts at the end and steps backward towards the start.

For loops

`Example01.py`

- Used to iterate through a sequence

```
numbers = [2, 3, 5, 7, 11]
for i in numbers:
    print(i)
```

```
2
3
5
7
11
```

```
name = "Andrew"
for c in name:
    print(c)
```

```
A
n
d
r
e
w
```

Example01.py

While loops

- Used to execute code when the end condition is unknown

```
name = "Andrew Luxton-Reilly"
i = 0
while name[i] != ' ':
    i += 1
print('Space is at position:', i)
```

```
Space is at position: 6
```

Example01.py

The Loop Else Clause

- The optional else clause runs only if the loop exits normally (not by break)

```
x = 1

while x < 3 :
    print(x)
    x = x + 1
else:
    print('hello')
```

```
1
2
hello
```

Exercise 4

- What is the output of the following code fragment?

```
number = 5
while number > 1:
  if number % 2 == 1:
    number = number * 3 + 1
  else:
    number = number // 2
  print(number, ",", end=" ")

else:
  print("EX3-END")
```

Loop Control Statements

break	Jumps out of the closest enclosing loop
continue	Jumps to the top of the closest enclosing loop
pass	Does nothing, empty statement placeholder

```
for letter in 'Python':
   if letter == 'h':
      break
   print('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
```

```
for letter in 'Python':
   if letter == 'h':
      continue
   print('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

Exercise 5

- What is the output of the following code fragment?

```
guess_str = input("Guess a number: ")
guess = int(guess_str)

number = 9
while 0 <= guess <= 100:
  if guess > number:
    print("Guessed too high!")
  elif guess < number:
    print("Guessed too low.")
  else:
    print("Bingo")
    break
  guess_str = input("Guess a number: ")
  guess = int(guess_str)
else:
  print("You quit early!")
```

Range

`Example01.py`

- Range is a special object in Python
  - Used to generate integer numbers within a range of values
  - Can iterate through the range

```
for x in range(0, 5):
    print(x)
```

```
0
1
2
3
4
```