

COMP120: Creative Computing: Tinkering

1: Computing Professionals

Learning Outcomes

- ▶ **Analyse** the role of computing professionals in the games industry
- ▶ Explain the role and basic functions of the IDE
- ▶ Produce some basic Python programs
- ▶ **Apply** pair programming practices to solve simple problems

Professional Roles

TwitterFall Activities

- ▶ Self-organise into small groups of 3-4
- ▶ Load a Twitter app, or login to Twitter on a PC
- ▶ Conduct research on the given topic
- ▶ Post a tweet when you find something interesting

- ▶ Please use the hashtag for the module (i.e., #*comp120*)
- ▶ Also please ensure you use the @ symbol to open and continue discussions

TwitterFall Activity #1

Answer the follow question:

“What do computing professionals do, *generally*?”

You have:

- ▶ 10 minutes to conduct research and tweet to *#comp120*
- ▶ 5 minutes to debrief

TwitterFall Activity #2

Answer the follow question:

“What do computing professionals do, *in games?*”

You have:

- ▶ 10 minutes to conduct research and tweet to *#comp120*
- ▶ 5 minutes to debrief

TwitterFall Activity #3

Answer the follow question:

“What career options are available to graduates with B.Sc. degrees in computing?”

You have:

- ▶ 10 minutes to conduct research and tweet to *#comp120*
- ▶ 5 minutes to debrief

Professional Development

Continuing Professional Development

- ▶ Games industry is fast-moving
- ▶ Learning does not end at school and university
- ▶ A goal of this course is to facilitate your development as self-regulated learners
- ▶ Gradually, more independence across each year of study
- ▶ This is a science degree, which means you will become a producer of knowledge, not just a consumer of knowledge!

Continuing Professional Development

- ▶ It isn't easy!
- ▶ Many of you will encounter programming anxiety
- ▶ Some will experience a sense of fear or a sense of hopelessness — it is more common than you think
- ▶ Some will need more support than others — this isn't a bad thing
- ▶ Everyone who puts in the time and effort will eventually achieve mastery

Professional Practice

Pair Programming

Pair programming is an agile software development technique in which two programmers work together at one workstation.

One, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in.

The two programmers switch roles frequently.

Pair Programming

Watch the video at:

<https://www.youtube.com/watch?v=ET3Q6zNK3Io>

(5 minutes)

Pair Programming

Review the guidelines at:

<http://www.pairprogramming.co.uk/>

(10 minutes)

Pair Programming

Watch the video at:

https://www.youtube.com/watch?v=ONnYCT_LJio

(5 minutes)

Pair Programming Challenge

- ▶ In pairs
- ▶ **Implement** the code excerpt
- ▶ **Fix** the errors in the code excerpt
- ▶ **Modify** the code excerpt to incorporate functions and arguments
- ▶ **Post** your solution to the `#comp120` slack channel

You can learn more about functions and arguments at:

<https://docs.python.org/3/tutorial/controlflow.html#defining-functions>

(20 minutes)

Pair Programming Challenge

The function:

```
def madlib()
```

Should become:

```
def madlib(name, pet, verb, snack)
```

Pair Programming Challenge

```
def madlib():
    name = 'Mike'
    pet = 'Spyro'
    verb = 'ate'
    snack = 'doughnuts'
    line1 = 'once upon a time,' + name + ' walked'
    line2 = ' with ' + pet + ', a trained dragon.'
    line3 = 'Suddenly, ' + pet + ' announced,'
    line4 = 'I really want some ' + snack + '!'
    line5 = name + ' complained. Where am I going to  ←
        get that?'
    line6 = 'Then ' + name + 'found a wizard's wand.'
    line 7 = 'With a wave of the wand, '
    line8 = pet + ' got ' + snack + '. '
    line9 = 'Perhaps surprisingly, ' + pet + ' ' +  ←
        verb + ' ' + snack
    print line1 + line2 + line3 + line4
    print line5 + line6 + line7 + line8 + line9
```

The PyCharm IDE

Integrated Development Environment (IDE)

- ▶ You *could* just write code in Notepad, but...
- ▶ An **Integrated Development Environment (IDE)** is an application providing several useful features for programmers, including:
 - ▶ A “run” button
 - ▶ Management of multi-file projects
 - ▶ Syntax highlighting
 - ▶ Autocompletion
 - ▶ Navigation
 - ▶ Language and API documentation
 - ▶ Debugging
 - ▶ Profiling
 - ▶ Version control

Setting up your own PC

► Python 2.7

- ▶ <https://www.python.org/>
- ▶ Python 2.7 is included with Mac OSX and most Linux distributions, but needs to be installed separately on Windows
- ▶ Python 2.x and Python 3.x are (slightly) different programming languages; we are using 2.x (for now)

► PyCharm

- ▶ <https://www.jetbrains.com/student/>
- ▶ Register with your `falmouth.ac.uk` email address to obtain PyCharm Professional Edition for free
- ▶ Runs on Windows, Mac and Linux
- ▶ Other Python IDEs are available

Getting started with PyCharm

- ▶ Create a new project (from the start-up wizard or from the File menu)
- ▶ We want a “Pure Python” project
- ▶ Right-click the project in the panel on the left, and choose “New → Python File”
- ▶ Write some code!
- ▶ First run: click “Run → Run...” and choose the Python file
- ▶ Subsequent runs: click the ▶ button

Basic Python programs

Your first Python program

```
print "Hello, world!"
```


Your second Python program

```
print "This is a very long line of code which had to  ←  
    be split to fit on the slide, but you should type  ←  
    it as a single line."  
print "This is the second line of code."
```

Assigning to variables

```
a = 10  
print a
```

Variable	Value
a	

Remember!

- ▶ A program is a **sequence of instructions**
- ▶ The Python interpreter executes the **first line** of your program, then the **second line**, and so on
- ▶ When it reaches the end of the file, it **stops**

Reassigning variables (1)

Socrative room code: FALCOMPED

```
a = 10  
b = 20  
b = a  
print a  
print b
```

Variable	Value
a	
b	

Reassigning variables (2)

Socrative room code: FALCOMPED

```
a = 10  
b = 20  
a = b  
print a  
print b
```

Variable	Value
a	
b	

Reassigning variables (3)

Socrative room code: FALCOMPED

```
big = 10  
small = 20  
big = small  
print big  
print small
```

Variable	Value
big	
small	

Reassigning variables (4)

Socrative room code: FALCOMPED

```
a = 10  
b = 20  
a = b  
b = a  
print a  
print b
```

Variable	Value
a	
b	

Reassigning variables (5)

Socrative room code: FALCOMPED

```
a = 10
b = 20
c = 30

a = b
b = c

print a
print b
print c
```

Variable	Value
a	
b	
c	

Reading input

```
print "Enter your name:"  
name = raw_input()  
  
print "Enter your age:"  
age = int(raw_input())  
  
print "Hello", name  
print "On your next birthday, you will be", age + 1, " ←  
years old"
```

- ▶ `raw_input()` reads a **string** as text from the command line
- ▶ `int(...)` converts a **string** into an **integer** (a number)

Conditionals (1)

Socrative room code: FALCOMPED

```
a = int(raw_input())  
b = 30  
  
if a < 15:  
    b = a  
  
print a  
print b
```

Variable	Value
a	
b	

Indentation

- ▶ Unlike many other programming languages, **indentation has meaning** in Python!
- ▶ Python uses indentation to denote the **block of code** inside a conditional, loop, function etc.
- ▶ PEP-8 recommends **4 spaces** for indentation
 - ▶ Some programmers use a tab character
 - ▶ **Never** mix tabs and spaces in the same file!
 - ▶ PyCharm inserts 4 spaces by default when you press the tab key; other IDEs and text editors can be configured to do this

Conditionals (2)

Socrative room code: FALCOMPED

```
a = int(raw_input())  
b = 0  
  
if a < 20:  
    b = a + 1  
elif a == 20:  
    b = a * 2  
else:  
    a = 20  
    b = 20  
  
print a  
print b
```

Variable	Value
a	
b	

Conditionals

An `if` statement can have:

- ▶ **Zero or more** `elif` clauses
- ▶ **An optional** `else` clause

In that order!

Mathematical operators

- ▶ + add
- ▶ - subtract
- ▶ * multiply
- ▶ / divide
- ▶ ** power

Order of operations: **BIDMAS**

- ▶ Brackets first
- ▶ Then Indices (powers)
- ▶ Then Division and Multiplication (left to right)
- ▶ Then Addition and Subtraction (left to right)

Comparison operators

- ▶ `<` less than
- ▶ `<=` less than or equal to
- ▶ `>` greater than
- ▶ `>=` greater than or equal to
- ▶ `==` equal to
- ▶ `!=` not equal to

Note the difference between `=` and `==`

- ▶ `a = b` means “make `a` be equal to `b`”
- ▶ `a == b` means “is `a` equal to `b`?”

For loops and ranges

```
for i in xrange(5):  
    print i
```

- ▶ `xrange(n)` is the **sequence** $0, 1, 2, \dots, n - 1$
- ▶ So `xrange(5)` is the **sequence** $0, 1, 2, 3, 4$
- ▶ Note: `xrange(n)` **does not include** n
- ▶ The `for` loop iterates through the items in a sequence **in order**
- ▶ Can also use `range` instead of `xrange`, but `range` is less efficient
 - ▶ Homework (advanced): what is the difference between `range` and `xrange`?

For loops (1)

Socrative room code: FALCOMPED

```
a = 0
b = 0

for i in xrange(5):
    a = i
    b = b + i

print a
print b
```

Variable	Value
a	
b	
i	

For loops (2)

Socrative room code: FALCOMPED

```
a = 0
b = 0

for i in xrange(10):
    if i < 3 or i > 7:
        a += i
    else:
        b += i

print a
print b
```

Variable	Value
a	
b	
i	

While loops

Socrative room code: FALCOMPED

The **while** loop keeps executing while the condition is **true**

```
a = 1

while a < 100:
    a = a * 2

print a
```

Variable	Value
a	

Looping forever

```
a = 1

while True:
    a = a * 2
    print a
```

Summary

We have seen some basic code constructions in Python

- ▶ `print` and `raw_input` for command-line input and output
- ▶ Variable assignment using `=`
- ▶ `if` statements for choosing whether or not to execute a block of code
- ▶ `for` loops to execute a block of code a specified number of times
- ▶ `while` loops to execute a block of code until a condition is no longer true

These are enough to write some simple programs, but you will see several more in coming weeks...