The UK's European university

University of Kent

COMP8270 / PROGRAMMING FOR ARTIFICIAL INTELLIGENCE

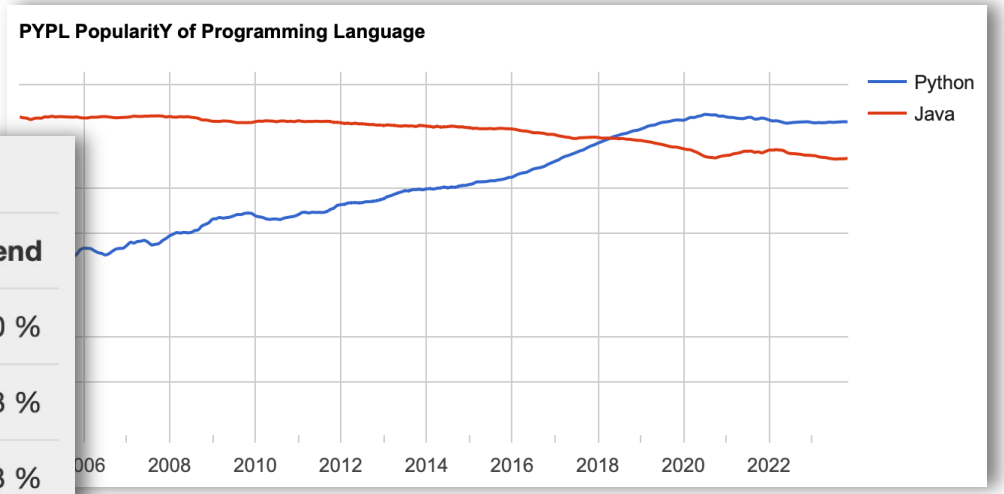
Fernando Otero
febo@kent.ac.uk
cs.kent.ac.uk/people/staff/febo

COMP8270:

- It provides a foundation for:
  - Python programming
  - writing code to load, manipulate and visualise data
  - use of libraries to complete tasks
  - application AI/ML techniques to process data and visualise results

- Builds on the concepts of COMP8810 Object Oriented Programming

PYPL PopularitY of Programming Language



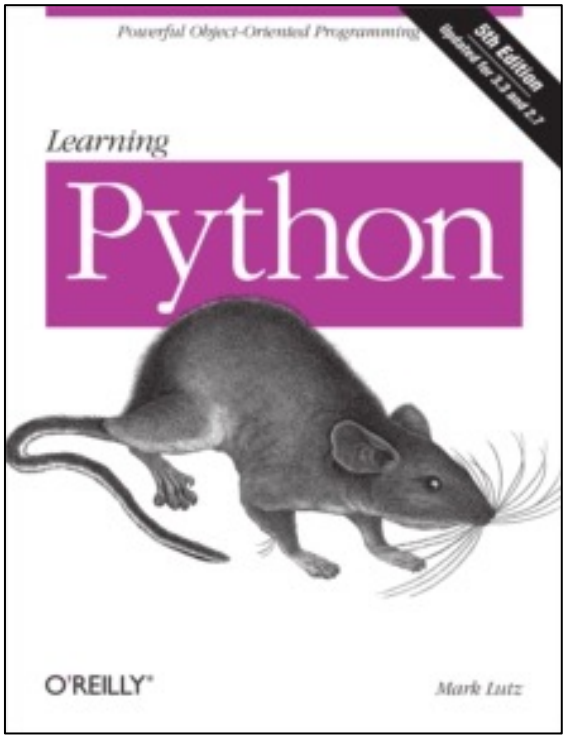
Worldwide, Nov 2023 :

Rank	Change	Language	Share	1-year trend
1		Python	27.99 %	+0.0 %
2		Java	15.91 %	-0.8 %
3		JavaScript	9.18 %	-0.3 %
4	↑	C/C++	6.76 %	+0.2 %
5	↓	C#	6.67 %	-0.3 %
6		PHP	4.86 %	-0.3 %
7		R	4.45 %	+0.4 %
8		TypeScript	2.95 %	+0.1 %
9	↑	Swift	2.7 %	+0.6 %
10	↓	Objective-C	2.32 %	+0.2 %

Textbook:

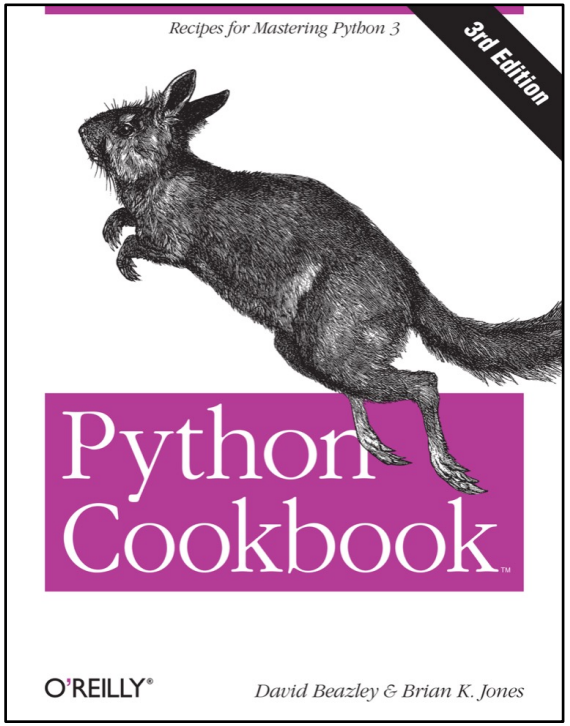
- Learning Python:
  - in-depth introduction
  - available in the library as an e-book

view book



Textbook:

- Python Cookbook:
  - "reference manual"
  - small/short individual topics with examples
  - available in the library as an e-book



view book

Moodle page:

- Lecture schedule
- Slides / Recordings
- Class Material
- Assessment information / submission

Structure:

- Three one-hour lectures per week
  - Monday, Tuesday and Thursday
- Two two-hour class per week
  - Check your timetable
- Support from class supervisor and lecturers

Assessment:

- Evaluate your ability to write programs

- 100% coursework (no exam)
  - Weekly assessed questions (20%)
  - Two (2 week) programming tasks (30% + 30%)
  - One time-limited programming task (20%) *
    - **Monday Week 18**

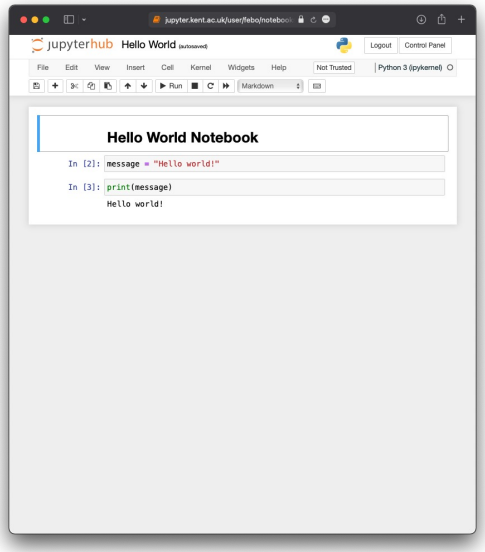
- The assessments progress in complexity
  - Practice makes you an **expert**!

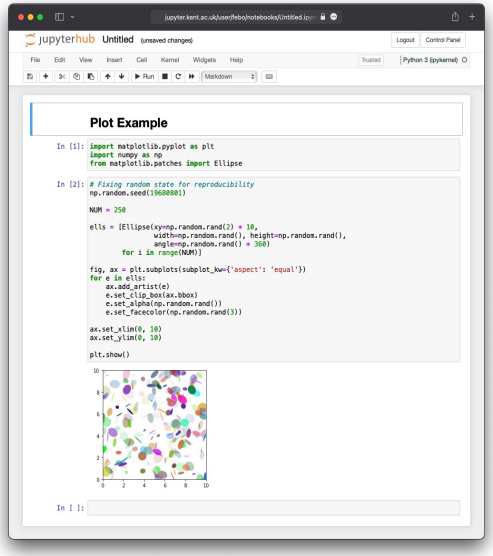
What you will be learning:

- Python syntax
- Collections: lists, sets, dictionaries
- List comprehension and slicing
- Functions and objects
- Python libraries:
  - Numpy
  - Pandas
  - scikit-learn
  - Keras

Environment:

- Jupyter Notebook:
  - web-based **interactive** development environment
  - easy input and output
  - industry standard





Key message:

- Learning does not happen only in lectures and classes
- you will need to practice:
  - write programs, make mistakes and learn from mistakes
  - go beyond what is presented in classes, use your curiosity to try different things
- Programming is fun!

introduction:

1. Python and Java

2. Basic syntax

Why Python?

- Programming language is usually a choice
  - Everyone has their on preferences
  - …in some cases there is a requirement

- (Some) characteristics that make Python great:
  1. Productivity:
     - less code to type (e.g., no `{ … }` or `;`), easier to read
     - dynamically typed (no need to specify types for variables)
     - easier to use (interpreted language), faster to get going
  2. Libraries:
     - very large collection of libraries, no need to start from scratch
     - complete libraries for artificial intelligence/machine learning

Static vs Dynamic Typing:

- Static typing:
  - Type checking at **compile** time
  - If type checking fails, program cannot run
  - Declare data types before use

- Dynamic typing:
  - Type checking at **run** time
  - Errors only happen when code is executed
  - Type is inferred from the declaration of the variable

Static vs Dynamic Typing:

```java
// java example
int n1 = 3;
int n2 = 5;
int result = n1 + n2;
```

```python
# python example
n1 = 3
n2 = 5
result = n1 + n2
```

Static vs Dynamic Typing:

- …but we still have strong type checking!

```
# python example
n1 = '3'
n2 = 5
result = n1 + n2

---------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_331965/3977660732.py in <module>
----> 1 n1 = n1 + n2

TypeError: can only concatenate str (not "int") to str
```

Compiled vs Interpreted:

- Java (compiled):
  - Source code
  - Compilation (Java Compiler)
  - Execution (Java Virtual Machine)

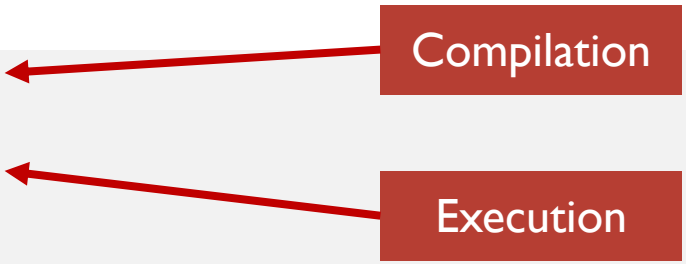
- Python (interpreted):
  - Source code
  - Execution (Python Virtual Machine)

“3 + 2” Example:

- Java (compiled):
  - Source code


```
public class Print5 {
  public static void main(String[] args) {
    System.out.println("3 + 2 = " + (3 + 2));
  }
}
```

  - On a terminal:




```
$ javac Print5.java
$ java Print5
3 + 2 = 5
```

“3 + 2” Example:

- Python (interpreted):
  - Source code

```
print("3 + 2 =", 3 + 2)
```

Source code

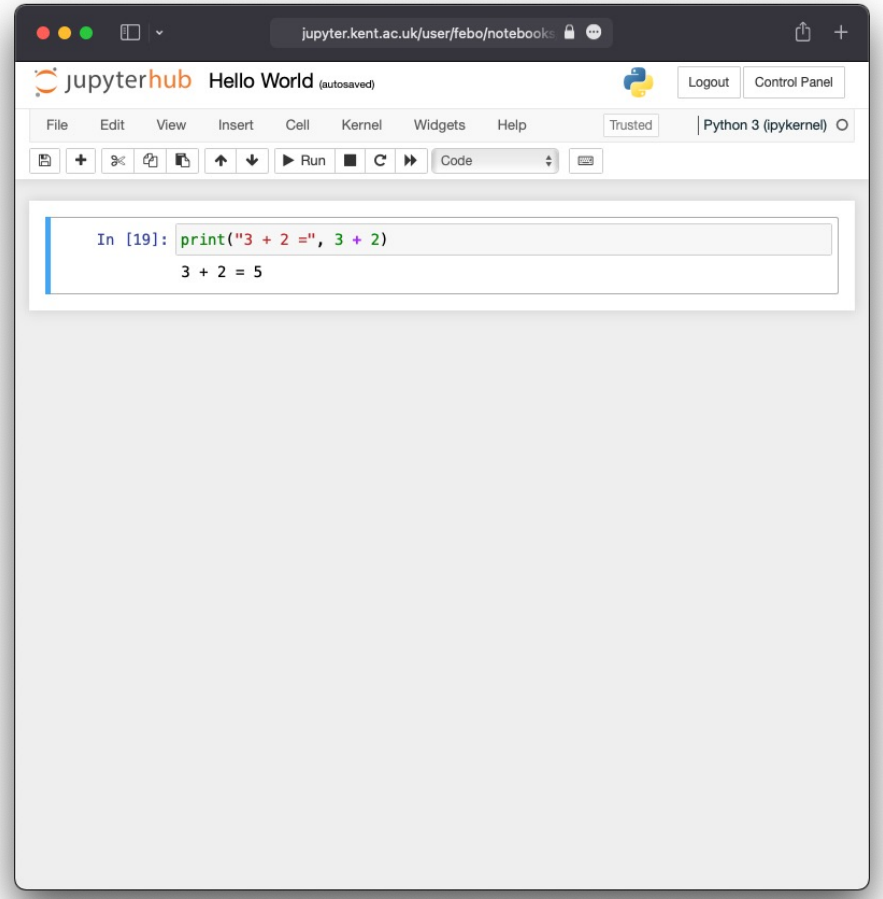
  - On a terminal:

```
$ python Print5.py
3 + 2 = 5
```

Execution

“3 + 2” Example:

- … or in a Jupyter Notebook:



introduction:

1. Python and Java

2. Basic syntax

Variables and Statements:

- Variables:
  - Should be lowercase, with words separated by underscores as necessary to improve readability
  - Variables are created the first time you assign it a value
- Statements do not need to be terminated by semi-colons

```
name = "Fernando”
staff_id = 1001
print(name)
print("Staff ID: ", staff_id)
```

Control Flow:

- **Indentation,** not braces:
  - Whitespace (tabs or spaces) are used to structure code instead of using braces

- A colon denotes the **start of an indented code** block after which all of the code must be indented by the same amount until the end of the block

Control Flow:

- if, elif, and else:

```
if x < 0:
    print("It's negative")
```

```
if x < 0:
    print("It's negative")
elif x == 0:
    print("Equal to zero")
else:
    print("It's positive")
```

Control Flow:

- for loops:

```
sequence = [1, 2, 3]
for value in sequence:
    print(value)
```

```
for i in range(6):
    print("Iteration #", i)
print("Outside for loop")
```

Control Flow:

- you can stop loops:

```
sequence = [1, 2, 3]
for value in sequence:
    if value == 2:
        break
    print(value)
```

Control Flow:

- while loops:

```
x = 256
total = 0
while x > 0:
    if total > 500:
        break
    total += x
    x = x // 2
```

- Use of bracket for the condition in loops is optional
  - if not needed, do not use them

▶ // is the floor divide (drops any fractional remainder)

Operators:

- ternary: *true-expr* if condition else *false-expr*

```
x = -5
print('negative' if x < 0 else 'positive')
```

- boolean:
  - `True` and `False`
  - `a and b`
  - `a or b`
  - `not a`

▶ More operators on "Numeric Type Basics" (Table 5-2, page 141)

Other bits:

- `range` function returns an iterator for a sequence of integer values:
  - range(*start*, *stop*, *step*)

Parameter	Description
start	integer number to start (default 0) [optional]
stop	integer number to stop (not included) [required]
step	integer number to increment (default 1) [optional]

```
range(10)
> sequence [0,1,2,3,4,5,6,7,8,9]
range(3, 20, 2)
> sequence [3,5,7,9,11,13,15,17,19]
```

Other bits:

- Comments:
  - Any text preceded by the hash mark # is ignored
  - Surround text in triple quotes

```
# This is a line of comment
message = "Python is cool"

# You can write more than one
# line of comments
print(message)

"""
Or use the mutiline string (triple quotes)
to add blocks of text for comments,
super useful to write long comments
"""
print("Back to coding")
```

Other bits:

- type casting functions:

```
s = '3.14159'
fval = float(s)
fval
> 3.14159
# cannot convert s to int directly
int(float(fval))
> 3
# converts fval to boolean
bool(fval)
> True
```

The Zen of Python:

```
> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Final remarks:

- We will practice everything from this lecture in the classes