



# PYTHON PROGRAMMING 101

**Peter Goodin**

Lead Data Science Instructor, GA Melbourne

Research Fellow, Neurorehabilitation, Royal Melbourne Hospital

Honorary Research Fellow, Florey Institute of Neuroscience & Mental Health



## LEARNING OBJECTIVES

- Discuss the history of Python & how it's used in different industries
- Define how Python compares to other programming languages
- Describe the benefits of a Python workflow when looking at data
- Demonstrate basic Python programming fundamentals to solve a real world problem
- Create a custom learning plan to build your Python skills after this workshop!

---

PYTHON PROGRAMMING 101

---

PRE-WORK



## PRE-WORK REVIEW

---

- Bring a [laptop with Anaconda](#) installed. Scroll to your operating system version and click on the install button for Anaconda with Python 2.7.
- We will be using [Jupyter Notebooks](#) as the main IDE for the workshop. If you have installed Anaconda, then you are ready to go!
- All the code, data and slides downloaded from:  
[https://github.com/petergoodin/ga\\_python101](https://github.com/petergoodin/ga_python101)

---

# PYTHON PROGRAMMING 101

---

# OPENING

# WHO AM I?

---

Peter Goodin

What I do: Lead Instructor – Data Science GA Melbourne,  
Neuroscience of Stroke / brain damage

What have I done: Ph.D Cognitive Neuroscience –  
Psychoneuroimmunology of Major Depressive Disorder,  
Consumer Neuroscience, Condom analysis.

What I'm interested in: Tech & data science for rehabilitation after  
brain damage.



petergoodin



@peter\_goodin

---

# ABOUT YOU!

---

- Before we dive in, let's talk a bit about you!
- Name
- What brings you to GA?
  - Current activities
  - Your Goals
- Fun fact about yourself!

---

 THE BIG PICTURE

---

- ▶ What we'll cover:
  - ▶ Why Python?
  - ▶ What can Python do for me?
  - ▶ Implementing Python into your workflow
  - ▶ Python Libraries
  - ▶ Programming (pseudocode and Python)
  - ▶ Dive into data with Python

---

 THE BIG PICTURE

---

- Why this topic matters:
  - Programming is a sought-after skill
  - Python has been gaining popularity (why? We'll find out!)
  
- Why this topic rocks:
  - Python opens up a door to a variety of opportunities, from data science to research



## WHAT IS PYTHON?

- Created by Guido Van Rossum in 1991
- Emphasises productivity and code readability
  - The language is easy to pick up and learn
  - This gentle learning curve brings makes it easier for many to contribute to production level code
  - Readable code means that almost anyone can pick up a piece of code and understand what it is doing

---

 THE ZEN OF PYTHON

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!

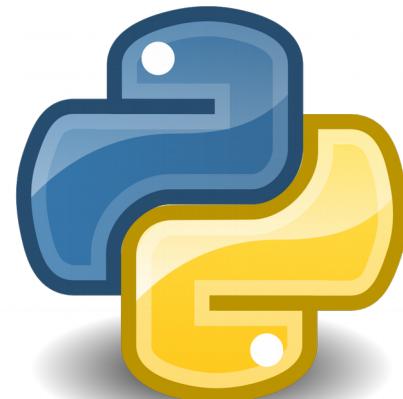
---



## SOME CHARACTERISTICS OF PYTHON

---

- Interpreted language: Code implementations execute instructions without having to translate them into machine-language instruction.
- In contrast, compiled code is executed by the computer's CPU (hardware)
- Interpreted code may run less quickly, but can be executed on multiple platforms without modification

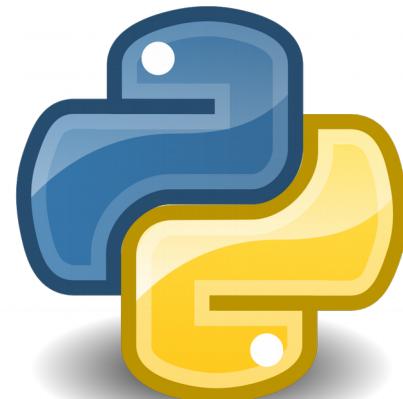


---

## SOME CHARACTERISTICS OF PYTHON

---

- Object-oriented (OO): Instead of concentrating on isolated "actions", object-orientation enables us to focus on "objects" that contain data & attributes.
- Objects have specific procedures, known as methods (code) that can access and modify the attributes of the object.
- OO makes it easier to reuse code in other programs

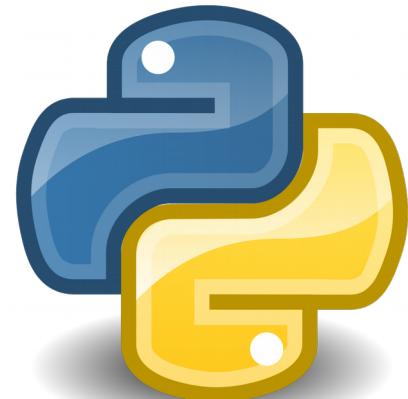


---

## SOME CHARACTERISTICS OF PYTHON

---

- High-level programming: This refers to the use of language similar to natural language makes it easy to use and automate.
- Related to the code readability mentioned earlier.

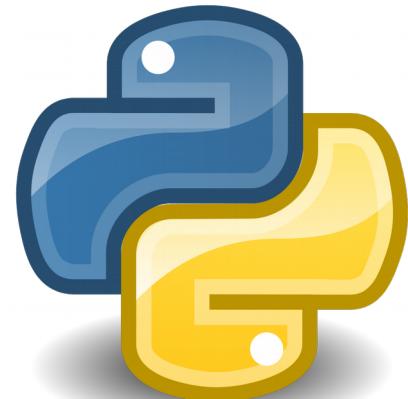


---

## SOME CHARACTERISTICS OF PYTHON

---

- Dynamic Typing: Variables are assigned their type by Python contextually.
- No need to pre-specify what kind of data a particular variable will have.
- Variable can be re-used (both good and bad), quick to make changes, generally faster to work with.



---

## WHY PYTHON?

---

Python is:

- Great for rapid prototyping and full-stack commercial applications.
- A modern, elegant, object-oriented language.
- Highly expressive, i.e., you can be more productive.
- Well documented and has an established and growing community.
- Comes with "batteries included" - in other words, Python has libraries that will help you do a ton of different tasks!



---



# WHY PYTHON?

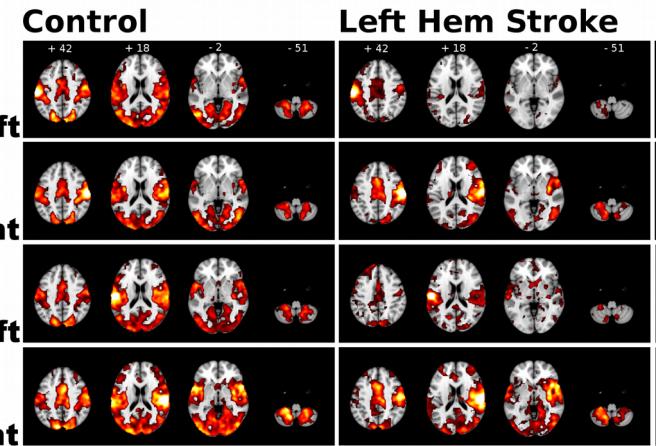
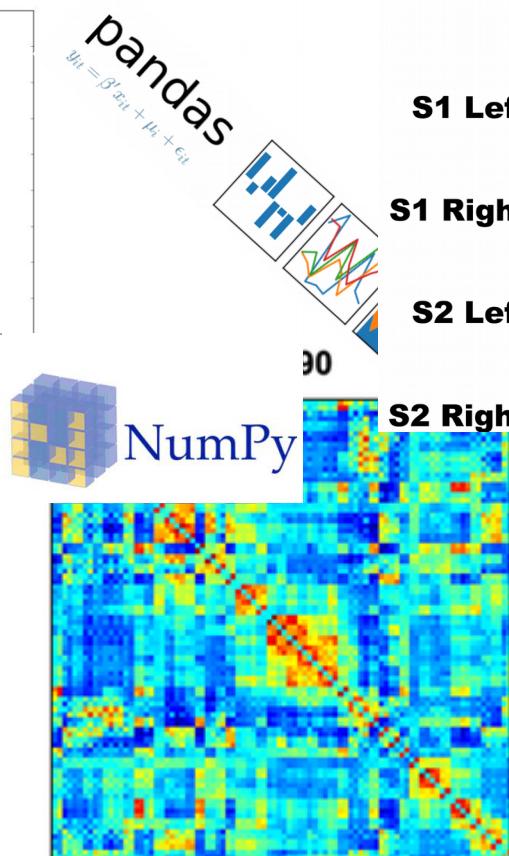
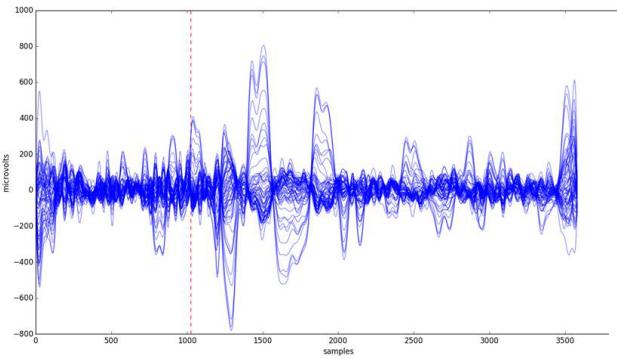
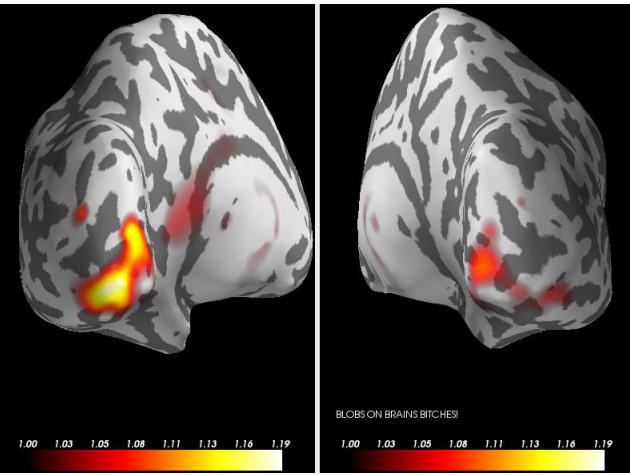
---

- Extremely fun to develop in :)
- Basically everything can be done with Python.
- If something can't be done, you can probably create an extension for it.
- Things can be done quickly!
  - For example a program that takes you weeks in C++, might take you a day in Python.



# DATA SCIENCE

## HOW DO I USE PYTHON?



PYTHON PROGRAMMING 101

# IMPLEMENTING A PYTHON WORKFLOW

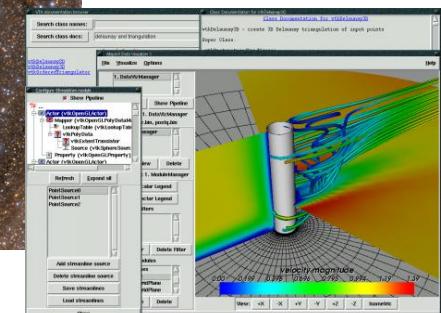
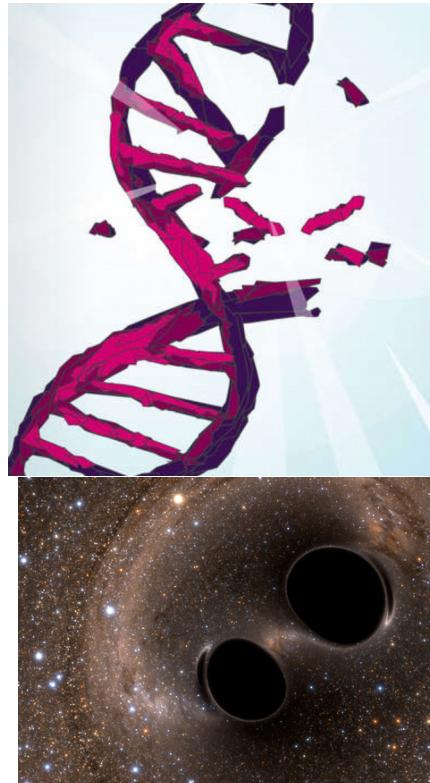
# WHERE IS PYTHON USED?

- Industry & Academia
- Art Middleware – Tools and Plugins
- Research
- Web Development and Web Applications
- Game Development
- Windows Applications
- You name it!



# CA EXAMPLES

- Industry
  - [Drug discovery](#)
  - [Financial services](#)
  - [Films and special effects](#)
- Academia
  - [Gravitational waves](#)
  - [Scientific visualisation](#)
  - [Biomolecule simulation](#)



# PYTHON Vs OTHER LANGUAGES

- Python is often compared to other interpreted languages, such as:
  - Java,
  - JavaScript,
  - Perl,
  - Tcl, or
  - Smalltalk.
- Comparisons to C++, Common Lisp and Scheme can also be enlightening.





# PYTHON PROGRAMMING

---

- Let us what a Python program looks like.
- Starting with the typical "Hello W o r l d !" program:
  - In essence, we are writing code to print the message "Hello World!" in the screen.



# PYTHON PROGRAMMING

Let's see the C++ version:

C++

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "Hello World!";
7     return 0;
8 }
```

- What do the components of the code do?



# PYTHON PROGRAMMING

What about Java?

Java

```
1 public class HelloWorld
2 {
3     public static void main (String[] args)
4     {
5         System.out.println("Hello, world!");
6     }
7 }
```

- Once again, it does exactly the same at the C++ version – Prints Hello, world! to the screen



# PYTHON PROGRAMMING

---

And in Python?



# PYTHON PROGRAMMING

And in Python?

```
1 | print("hello world")
```

Python



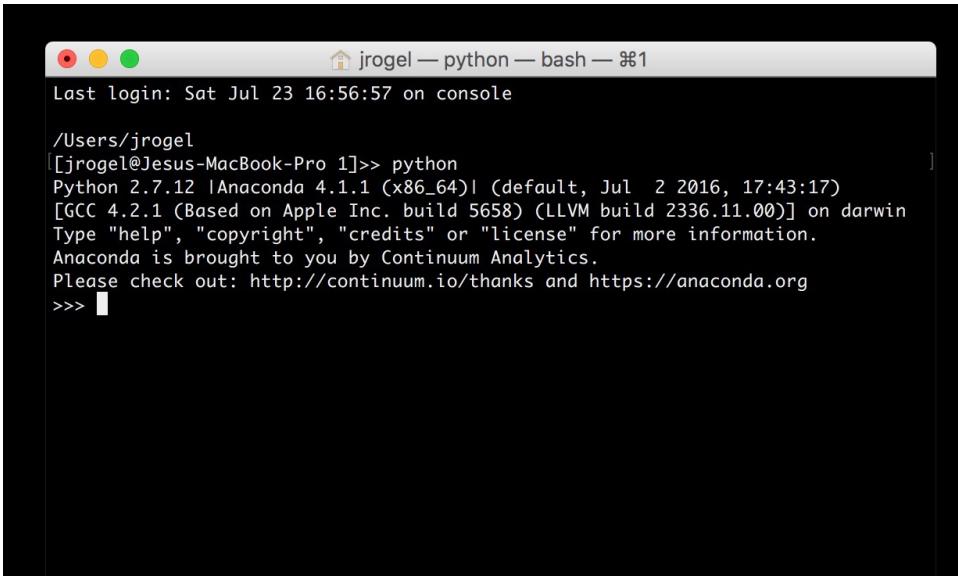
# PYTHON: INTERACTIVE SHELLS Vs. SCRIPTS

- In our “Hello World!” python program, we are assuming that we are using an interactive shell,
- In other words, we are writing code that is executed immediately by the Python interpreter.
- We are able to "interact" with the results of the commands we pass. We can do this using a:
  - Python shell
  - iPython shell
  - Jupyter notebook



# PYTHON SHELL

- A python shell is similar to a Command Line Terminal and it can be launched by typing:
- “**python**”



A screenshot of a macOS terminal window titled "jrogel — python — bash — %1". The window shows the following text:

```
Last login: Sat Jul 23 16:56:57 on console  
/Users/jrogel  
[jrogel@Jesus-MacBook-Pro 1]>>> python  
Python 2.7.12 |Anaconda 4.1.1 (x86_64)| (default, Jul 2 2016, 17:43:17)  
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2336.11.00)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
Anaconda is brought to you by Continuum Analytics.  
Please check out: http://continuum.io/thanks and https://anaconda.org  
>>> █
```



# DA iPYTHON SHELL

- A python shell is more interesting than a plain terminal, providing syntax colouring and shortcuts to interact with our code. It can be launched with:
- “**ipython**”

```
jrogel — python ▾ python.app /Applications/anaconda/bin/ipython — bash — ⌘1

/Users/jrogel
[jrogel@Jesus-MacBook-Pro 5] >>> ipython
Python 2.7.12 |Anaconda 4.1.1 (x86_64)| (default, Jul 2 2016, 17:43:17)
Type "copyright", "credits" or "license" for more information.

IPython 4.2.0 -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help        -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

[In [1]: 1+1
Out[1]: 2

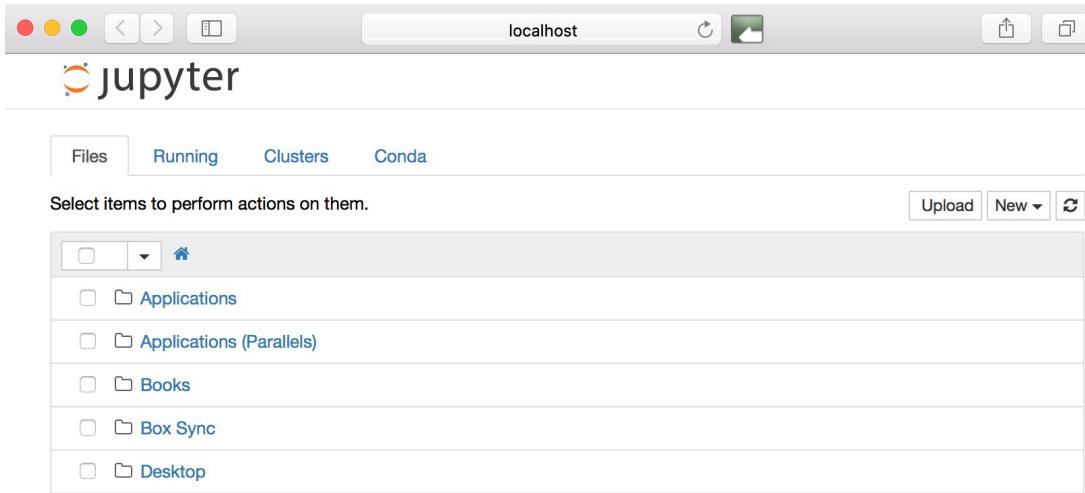
[In [2]: print("Hello World!")
Hello World!

In [3]: ]
```



# JUPYTER NOTEBOOK

- A Jupyter notebook is a web interface that lets us use formatting along side our code. It is extremely common and very useful! You can launch it by typing:
- **“jupyter notebook”**





## PYTHON: INTERACTIVE SHELLS Vs. SCRIPTS

- › Sometimes we do not need to interact with our Python code.
- › Instead, we may want to execute a program and simply get results.

In those cases, we need to create a Python script.

- › To do so, we can use a text editor of our choice and save the code in a file with extension “.py”.





## SCRIPT EDITORS AND IDE

- Some common plain text editors include:
  - Atom
  - NotePad (Win), NotePad++ (Win)
  - TextEdit (Mac), TextWrangler (Mac).
  - Nano, Gedit (Linux, Mac)



---

## SCRIPT EDITORS AND IDE

---

- Integrated development environments (IDEs) provide comprehensive facilities for computer programmers involved in software development.
  - [PyCharm](#)
  - Eclipse with [PyDev](#)
  - [Atom](#)
  - Spyder (included in Anaconda)



## SIMPLE PYTHON SCRIPT

- A barebones script for the "Hello World!" program (saved to a file called `hi.py`) looks like this:

```
1 | print("hello world")
```

Python

- To run the script by passing it as a command to the Python interpreter we need to write:

```
 $\nu_e, \nu_\mu, \nu_\tau$   
› python hi.py
```

## Demo

---

# VISUALISING DATA

- Is it important?
- When is it needed?
- What does it add?
- What tools do you use?

# INSTRUCTIONS

- We recommend using a Jupyter notebook for this demo.
1. Save the file called Python\_101\_DemoBokeh.ipynb in a known location in your file system
  2. Open Jupyter: in a terminal type: `jupyter notebook`
  3. Navigate to the folder where you saved your file in step 1.
  4. Click on the name of the file
  5. Voilà, you are ready to follow the demo



---

## Guided Practise

---

# COMMON PYTHON PACKAGES & DATA TYPES

# DATA PACKAGES

- Libraries of code written to solve particular set of problems
  - Can be installed with: **pip install <package name>**
  - or **conda install <package name>**
- 
- Ever used Excel? How do yo fancy working with data structured in a similar way, but better graphics and less hassle? Try **pandas**
  - Does your application require the use of advanced mathematical or numerical operations using arrays, vectors or matrices? Try **SciPy** (scientific python) and **NumPy** (numerical python)



## PACKAGES

- Libraries of code written to solve particular set of problems
- Can be installed with: **pip install <package name>**
- or **conda install <package name>**
- Does your application require the use of advanced mathematical or numerical operations using arrays, vectors or matrices? Try **SciPy** (scientific python) and **NumPy** (numerical python)



---

# DATA PACKAGES

- Are you interested in using python in a data science workflow to exploit machine learning in your applications?  
Look no further than **Scikit-learn**
- Are you tired of boring-looking charts? Are you frustrated looking for the right menu to move a label in your plot? Take a look at the visuals offered by **matplotlib**



---

## PACKAGES

---

- Is your boss asking about significance testing and confidence intervals? Are you interested in descriptive statistics, statistical tests, or plotting functions? Well [statsmodels](#) offers you that and more.
- All the data you require is available freely on the web but there is no download button and you need to scrape the website? You can extract data from HTML using [Beautiful Soup](#) and automate web browsing using [Selenium!](#)



# INSTRUCTIONS



- We recommend using a Jupyter notebook for this practice.
- 
1. Open Jupyter: in a terminal type: `jupyter notebook`
  2. Navigate to an appropriate folder where your work will be saved
  3. On the top-right-hand-side click in the button called "New" and select "Python 2" or "Root" (depending on your installation of Python)
  4. Voilà, you are ready to type the commands we will cover

# DATA IMPORTING A MODULE

- We need to import the functionality of packages and modules before we can use them
- Here we import the “math” module to use mathematical functions:



Python

```
1 import math
2 x = math.cos(2 * math.pi)
3 print(x)

4

5 from math import *
6
7 log(10)
8
9 log(10,2)
```

# DATA TYPES, VARIABLES, ASSIGNMENT

- Like any other programming language, we need to use types and variables and be able to assign values to them



Python

```
1 # variable assignments
2 x = 1.0
3 my_variable = 12.2
4 type(x)
5
6 y = 1
7 type(y)
8
9 b1 = True
10 type(b1)
11
12 s = "String"
13 type(s)
```

## YOUR TURN

- Try the following in your Jupyter Notebook:

Python

```
1 import types
2 print(dir(types))

3
4 1+2, 1-2, 1*2, 1/2

5
6 1.0+2.0, 1.0-2.0, 1.0*2.0, 1.0/2.0

7
8 # Comment

9
10 # Comparison: >, <, <=, <=, ==
11 2 > 1

12
13 # Testing for equality
14 2 == 2
```



# CA LISTS ASSEMBLY

- Lists are collections of objects
- They can be changed



Python

```
1 l = [1,2,3,4]
2 print(type(l))
3 print(l)
4 print(l)
5 print(l[1:3])
6 print(l[::-2])
7
8 # Python starts counting from 0
9 print(l[0])
```

# DATA TUPLES

- Tuples are very similar to lists, but
- they cannot be changed



Python

```
1 point = (10, 20)
2 print(point, type(point))
3
4 x, y = point
5 print("x =", x)
6 print("y =", y)
```

# DATA DICTIONARIES

- Dictionaries combine keys with values in pairs
- Like in a dictionary, you can search the keys to obtain their corresponding value



Python

```
1 | params = {"parameter1" : 1.0, "parameter2" : 2.0,  
2 |     "parameter3" : 3.0,}  
3 | print(type(params))  
4 | print(params)
```

# CLASS ASSEMBLY

- ▶ In Object Oriented Programming, a **class** is a template definition of the methods and variables in a particular kind of object
- ▶ In other words, an **object** is a specific instance of a **class**
- ▶ The **object** contains actual values instead of variables.

## Class Definition

```
Circle
-radius:double=1.0
-color:String="red"
+getRadius():double
+getColor():String
+getArea():double
```

## Instances

<u>c1:Circle</u>
-radius=2.0
-color="blue"
+getRadius()
+getColor()
+getArea()

<u>c2:Circle</u>
-radius=2.0
-color="red"
+getRadius()
+getColor()
+getArea()

<u>c3:Circle</u>
-radius=1.0
-color="red"
+getRadius()
+getColor()
+getArea()

---

## INDEPENDENT PRACTICE

---

THIS IS GOING TO  
BUILD TO  
SOMETHING

## PSEUDO CODE

- Pseudocode allows us to represent a program concisely.
- The only thing you need is a statement to show where you are starting and where you are ending a program.
- Calculate and print the average of three numbers: 5, 10, and 15.

```
1 Start
2     num1 = 5
3     num2 = 10
4     num3 = 15
5     sum = num1 + num2 + num3
6     average = sum/3.0
7     print average
8 End
```



# ACTIVITY: WRITE YOUR OWN (PSEUDO)CODE

## DIRECTIONS



EXERCISE

1. Create some pseudocode for the following tasks
  1. Create a short script that will calculate the area circle with radius r.
  2. Calculate and print the square of a number. If the number is larger than 10 also calculate the cube.
  3. List the letters in the sentence "Python is awesome"

## DELIVERABLE

Pseudocode explaining the necessary steps to achieve one of the tasks above

---

## INTRODUCTION

---

# PYTHON PROGRAMMING FUNDAMENTALS

## INSTRUCTIONS



- We recommend using a Jupyter notebook for this practice.
- 
1. Open Jupyter: in a terminal type: **`jupyter notebook`**
  2. Navigate to an appropriate folder where your work will be saved
  3. On the top-right-hand-side click in the button called "New" and select **"Python 2"** or **"Root"** (depending on your installation of Python)
  4. Voilà, you are ready to type the commands we will cover



# PYTHON PROGRAMMING FUNDAMENTALS

---

- Understanding core programming concepts and why they are used is just as important as knowing how to write code.
- Before we start, we'll review some basic programming concepts:
- Variables: A symbolic name that stores a value (some specific piece of information). Variables have different types.
- For example: **r = 3**





# PYTHON PROGRAMMING FUNDAMENTALS

- Control Structures: A block of programming that analyses variables and chooses a direction in which to go based on given parameters.
- The term flow control details the direction the program takes (which way program control “flows”). It determines how a computer will respond when given certain conditions and parameters. Some typical structures include:
  - If statement
  - For loop
  - Functions





# PYTHON PROGRAMMING FUNDAMENTALS

- Data Structures: Data structure is a particular way of storing and organising data in a computer so that it can be used efficiently. Some examples include:
  - Lists ← Native
  - Tuples ← Native
  - Dictionaries ← Native
  - Arrays ← Numpy
  - Matrices ← Numpy
  - Dataframes ← Pandas





# PYTHON PROGRAMMING FUNDAMENTALS

---

- Syntax: The syntax of a programming language is the set of rules that define the combinations of symbols that are considered to be correctly structured programs in that language
- The interrelationship of all of these elements make it possible for us to write programs to implement algorithms and solve problems



## GA | FERAL ASSEMBLY

An `if` statement is a conditional structure that, if proved true, performs a function or displays information.

Think of this as a decision that moves the flow of your program depending on the answer to a TRUE-FALSE question.

In pseudocode:

```
1 | IF a person is older than 18  
2 | THEN they can drive  
3 | ELSE they cannot drive
```

In Python we can write:

```
1 | if age_person > 18:  
2 |     return "They can drive"  
3 | else:  
4 |     return "They cannot drive"
```

Python

Another example:

Python

```
1 A = 10
2 B = 100
3 if A>B:
4     print("A is larger than B")
5 elif A==B:
6     print("A is equal to B")
7 else:
8     print("A is smaller than B")
```

# FOR LOOP

A loop statement in programming performs a predefined set of instructions or tasks while or until a predetermined condition is met.

Think of this as a repetitive action that has to be performed until further notice.

In pseudocode:

```
1 | FOR each user of a service in a list  
2 |     PRINT greet them
```

In Python we can write:

```
1 | users = ["Jeff", "Jay", "Theresa"]  
2 |  
3 | for user in users:  
4 |     print("Hello %s" % user)
```

Python

# FOR LOOP

Tip: When creating a for loop, make sure it's condition will always be met to help prevent an endless loop!

Let us see other examples. Can you explain what the program is doing?

Python

```
1 | for x in [1,2,3]:  
2 |     print(x)  
3 |  
4 | for key, value in params.items():  
5 |     print(key + " = " + str(value))
```

## LIST COMPREHENSION

List comprehension is an elegant way to define and create list in Python. It uses a for loop inside the definition of the list itself.

Let's take a look at one, and see if you can figure out what is happening:

```
1 | l1 = [x**2 for x in range(0,5)]
```

Python

---



# FUNCTIONS

---

A function is a group of instructions used by programming languages to return a single result or a set of results.

Functions are a convenient way to divide our code into useful blocks, providing us with order, making the code more readable and reusable.



# FUNCTIONS

Here is how you define a function in Python:

Python

```
1 def function_name(input1, input2...):  
2     1st block of instructions  
3     2nd block of instructions  
4     ...
```

Let's define a function that returns the square of the input value:

Python

```
1 def square(x):  
2     """  
3         Return the square of x.  
4     """  
5     return x ** 2
```

# GA FUNCTIONS

We can call this function as follows:

Python

```
1 | var1 = 7  
2 |  
3 | var2 = square(var1)  
4 |  
5 | print(var2)
```



---

DEMO

---

# WRITING PROGRAMS IN PYTHON



# WRITING PROGRAMS IN PYTHON

- ▶ Python is an interpreted language, which means you can run the program as soon as you make changes to the file.
- ▶ This makes iterating, revising, and troubleshooting programs is much quicker than many other languages.
- ▶ Let us create a complete program that will calculate the area circle with radius r.

Python

```
1 # We are importing the value of pi from
2 # that module - Easy to read, right?
3 from math import pi
4
5 def circ_area(r):
6     return pi * r**2
7
8 r = 3
9 area = circ_area(r)
10 print(area)
```



# ACTIVITY: WRITE YOUR OWN CODE!

## DIRECTIONS

A teal circle with a white wedge at the bottom right, resembling a slice of pizza or a pie chart.

EXERCISE

1. Translate the your pseudocode into Python code!
  1. Create a short script that will calculate the area circle with radius r.
  2. Calculate and print the square of a number. If the number is larger than 10 also calculate the cube.
  3. List the letters in the sentence "Python is awesome"

## DELIVERABLE

Python code that executes the tasks above

---

## GUIDED PRACTISE

---

LET'S  
EXPLORE  
SOME DATA

## INSTRUCTIONS



- Let's open a new Jupyter notebook for this practice.
  - We will work in pairs.
1. Save the file called **Python101\_Part2\_GuidedPractice.ipynb** in a known location in your file system
  2. Start Jupyter (you know how now) and navigate to the location where you saved the file
  3. Open the file by clicking on the name
  4. Voilà, you can start the Guided Practice

---

INDEPENDENT PRACTISE

---

YOUR TURN!

## OPTION 1



- Let's open a Jupyter notebook for this practice.
- 
1. Save the file called **Python101\_Part2\_IndPractice.ipynb** in a known location in your file system
  2. Start Jupyter (you know how now) and navigate to the location where you saved the file
  3. Open the file by clicking on the name
  4. Voilà, you can start the Independent Practice

---

PYTHON PROGRAMMING 101

---

# CONCLUSION

---

 REVIEW & RECAP

---

In this workshop, we've covered the following topics:

- Python as a popular, flexible programming language
- Python has applications in many different areas
- Python is particularly great for data manipulation
- Python programming basics include: types, variables, functions, and more!

---

 TAKEAWAYS

---

# LEARNING PLAN

Evaluate your python programming skills! How confident are you with:

- Python Syntax
- Programming Fundamentals
- Data Analysis

---

 TAKEAWAYS

---

# WHAT SHOULD YOU DO NEXT?

For beginner programmers:

- Familiarise yourself with the language with an [Introduction to Python](#)
- Go through [Learn Python the hard way](#)
- And [Automate the boring stuff](#)

---

 TAKEAWAYS

---

# WHAT SHOULD YOU DO NEXT?

For existing programmers who are new to Python, try these:

- Read the information in [Moving to Python from other languages](#)
- [Python for Java Developers](#)
- [Python for MATLAB users](#)

---

 TAKEAWAYS

---

# WANT MORE?

General Assembly offers courses that use Python!

Check out our:

- [Part-time Data Science Course](#)
- [Data Science Immersive Course](#)

---

# PYTHON PROGRAMMING 101

---

# Q&A

---

PYTHON PROGRAMMING 101

---

# EXIT TICKETS

DON'T FORGET TO FILL OUT YOUR EXIT TICKET