Introduction To Programming

What is Programming

Programming is the process of writing a precise set of instructions to be interpreted by a computer

Programming gives you the power to control what a computer does

Why Python?

1) Simpler syntax than other languages
2) Code is short and easy to read
3) It is easy to learn
4) Libraries available to do almost anything
5) High level
6) Not too high level
7) Fun to use!

Why not Python?

1) Python can be slow

2) Python isn't great at dealing with low level hardware

Installation

Windows:

- Open “python.org/getit” in your favourite web browser
- Download the python installer
- Click it and press next a few times

Linux:

- It should already be installed
- Open up a terminal and type python -v to see what version you are using

Editors

Windows:
- Notepad++
- IDLE

Linux:
- VIM
- Gedit

OSX:
- Textmate (~€50)
- Textwrangler

All:
- Sublime Text 2

Integrated Development Environments

Pycharm (free -> €180)

Aptana Studio

Python Tools for Visual Studio

Spyder

Using the Interpreter

Type the following commands into the interactive terminal:

```
print(“Hello World!”)

4 * 5

a = 4

b = 5

a * b
```

Basics

Syntax – The way in which the language is written

Immutable – An object that can't be changed

Literals – A literal is a literal thing like a string or a number

Comments – Used to give more information about a piece of code

```
# A Comment is like this
```

Numbers – Numbers are either intergers, (10293, -5838) or floats, (1.345, -553.455, 34.21e-45)

Strings – A sequence of characters enclosed by double or single quotes

Variables – A named way for accessing memory where something is stored, accessed through identifiers

Identifiers – These just provide a system to call functions and variables by name

Iterable – An object that you can iterate over

Mathematical Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer division (Python 3+)
%	Modulo division
**	Power
=	Assignment Operator

Logical Operators

and	Check if 2 things are true
or	Check if at least 1 of 2 things is True
not	Check if something is not true
==	Check if 2 things are equal
!=	Check if 2 things are not equal
<	Check if something is less than something else
<=	Check if something is less than or equal to something else
>	Check if something is greater than something else
>=	Check if something is greater than or equal to something else
in	Check if something is in something else
is	Check if something is something else

Keywords

and del from not while
as elif global or with
assert else if pass yield
break except import print
class exec in raise
continue finally is return
def for lambda try

Flow Control – if, elif, else

These are the most common flow control keywords.

They allow you to use conditions to decide which blocks of code are executed.

The general Format is

```
if <condition>:
    do something
elif <different condition>:
    do something else
elif <another different condition>:
    do another different thing
else:
    make one last ditch attempt to do somthing
```

Flow Control – for

For allows you to loop over an iterable until that iterable is consumed or until a condition is reached

```
# print the numbers 0 - 9
for i in range(10):
    print(i)

# print the letters 'a' 'b' 'c' and 'd'
for l in ['a', 'b', 'c', 'd']:
    print(l)
```

Flow Control – while

Do something until a block is no longer true

```
a = 100
# reduce a by 1 until a is 0
while a > 0:
    a -= 1

while True:
    # Loop forever
    pass
```

Flow Control – break

This allows you to break out of the innermost for or while loop when it is called

```
for i in range(5):
    for j in range(2,6):
        if j == 3:
            break
    print('{0} {1}'.format(i, j)
```

Flow Control – continue

This allows you to ignore the rest of the current run through the for or while loop

```
for i in range(100):
    if i%2 == 0:
        continue
    print(i)
```

Functions

```
def hello():
    '''Prints out “Hello World!” '''
    print('Hello World!')

hello()
```

Functions - Params

```
def age(year, month, day):
    '''Calculates a person's age from their date of birth'''
    from datetime import datetime as dt
    now = dt.now()
    birth = dt(year, month, day)
    years = (birth-now).days//365.25
    print(“You are {0} years old”.format(years))
```

Functions - Defaults

```
def join(sequence, sep=', ')
    ''' Joins a sequence together '''
    print(sep.join(sequence))

join([1, 2, 3, 4]) # Prints: 1, 2, 3, 4

join([1, 2, 3, 4], '.') # Prints: 1.2.3.4
```

Functions - Return

```
def mult(x, y):
    ''' Multiplies x and y together '''
    return x*y

def floor_remainder(x, y):
    ''' Returns the integer division and remainder of x and y '''
    return x//y, x%y

m = mult(8, 8)
d, r = floor_remainder(64, 5)
```

Packages

Python has a huge standard library containing most of what you might need.

To use the standard library you must import the functions or objects you want.

There are a few ways of doing this though with provides great flexibility

Packages

```
import math
math.sqrt(math.pi) # prints out 1.7724....
import math as ma
ma.sqrt(ma.pi) # prints out 1.7724...
from math import pi, sqrt
sqrt(pi) # prints out 1.7724...
from math import *
sqrt(pi) # prints out 1.7724...
```

Packages

Information on the standard library:
http://docs.python.org/3/library/index.html

Information on thousands of user made libraries:
https://pypi.python.org/pypi

Data Structures - List

- List `>>> l = []`
  - Ordered collection of objects
  - Can grow or shrink in size
  - Can store any object
  - Functions:
    - .append(<object>) add an object to a list
    - .index(<value>) find index of first matching value
    - .sort() sort the list
    - .copy() copies the list
    - .clear() sets all objects in the list to None
    - .remove(<value>) removes first incidence of value
    - .insert(<index>, <object>) insert the object at the index
    - .pop(<index>) remove the object at the index and return its value
    - .count(<value>) counts the number of occurrences of a value

Data Structures – List Indexes and Slices

- List `>>> l = []`
  - To access an element in a list:

    ```
    l[0] # Returns the first element of the list
    l[-1] # Returns the last element of the list
    ```

  - To create a sublist:

    ```
    l[1:4] # Create a sublist of the second third and fourth elements of the list
    ```

  - To copy a list:

    ```
    l_2 = l[:]
    ```

Data Structures - Tuple

- Tuple `>>> t = ()`
  - Ordered collection of objects
  - Fixed size, hashable, immutable
  - Can store any object
  - Functions:
    - .index(<value>) find index of first matching value
    - .count(<value>) counts the number of occurrences of a value

Data Structures - Set

- Set `>>> s = set()` **OR** `>>> s = {<object_1>, <object_2>}`
  - Unordered collection of objects
  - Works like a set from mathematical set theory
  - Can store any object
  - Can only store one copy of each object
  - Functions:
    - .add(<object>) add an object to a list
    - .copy() copies the list
    - .clear() sets all objects in the list to None
    - .remove(<value>) removes value from the set
    - .discard(<value>) removes value from the set if it exists
    - .pop(<index>) remove the object at the index and return its value

Data Structures - String

- String `>>> s = “” OR s = ''`
  - Ordered collection of characters
  - Hashable, Immutable
  - Functions:
    - .join(<iterable>) join all of the objects in the iterable separated by this string
    - .split(<string>) return a list of the string split by the given string
    - .upper() convert string to uppercase
    - .lower() convert string to lowercase
    - .find(<string>) find a matching string within another string
    - .count() count the number of characters in the string

Data Structures - Dictionary

- Dict `>>> s = {}`
  - Unordered collection of key value pairs
  - Functions:
    - .copy() copies the dictionary
    - .clear() remove all items from the dictionary
    - .remove(<value>) removes first incidence of value
    - .pop(<key>) remove the object at the key and return its value
    - .keys() returns a set of keys
    - .values() returns a list of values
    - .get(<key>) get the value for a key
    - .setdefault(<value>) set a default value for when a key doesn't exists and create then key with the default value when .get(<key>) is called

I/O

```
var = input('Asking Text Here: ')
f = open('File Path Here', 'w')
f.write(var)
f.close()
f = open('File Path Here')
text = f.read()
print(text)
```

Resources

http://www.reddit.com/r/python

Any Questions?