Introduction to Python

Basic Syntax

Datatypes I: Integers and Floats

Topics

1) Python interpreters
2) Python Scripts
3) Printing with print()
4) Basic Built-In Number Types
    a) Integers
    b) Floats

Python Interpreter

The most basic way to execute Python code is line by line within the *Python interpreter*.

Assuming that Python is already installed, typing "python" at the command prompt (Terminal on Mac OS X and Unix/Linux systems, Command Prompt in Windows):

```
(base) L-C1MWF3UYJ1WK-134540:~ 134540$ python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python Interpreter

With Python interpreter, we can execute Python code line by line.

```
>>> 1 + 1
2
>>> x = 5

>>> x * 3
15
```

IPython Interpreter

It is recommended that you install Anaconda distribution which includes Python and many useful packages for scientific computing and data science.

Bundled with Anaconda is the IPython interpreter.

Type "ipython" on your terminal(Mac) or command prompt(Windows).

```
(base) L-C1MWF3UYJ1WK-134540:~ 134540$ ipython
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

IPython Interpreter

The main difference between the Python interpreter and the enhanced IPython interpreter lies in the command prompt:

Python uses >>> by default, while IPython uses numbered commands.

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

In[2]: x = 5
In[3]: x * 3
Out[3]: 15
```

Python Scripts

Running Python snippets line by line is useful in some cases, but for more complicated programs it is more convenient to save code to file, and execute it all at once.

By convention, Python scripts are saved in files with a *.py* extension.

Scripts are written using an IDE(Integrated Development Environment). We will initially use an online IDE (repl.it, login with your Google account) to learn the basics of Python.

Then later, when we start writing games, we will use a popular local IDE called Visual Studio Code.

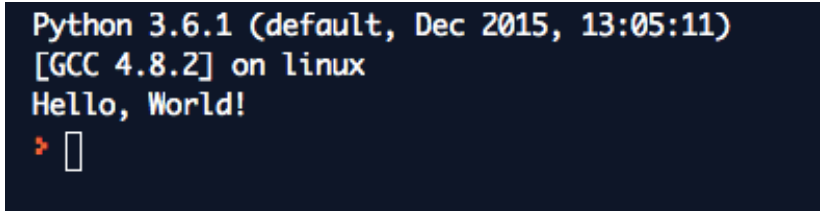
Our First Program

```
print("Hello, World!")
```

1) The print() function prints messages on the console.

2) Characters enclosed in quotes(single or double) forms a **literal string**.

3) The console output string does not include the quotes.



```
print(*objects, sep=‘ ’, end=‘\n’)
```

any number of **positional arguments**

specify separator character; defaults to space

specify end character; defaults to newline

```
print("hello")
print("hello", "Mike")
print("hello", "Mike", "how are you?", sep=”,")
print("home", "user", "documents", sep="/")
```

Output:

hello

hello Mike

hello,Mike,how are you?

home/user/documents

```
print(*objects, sep=‘ ’, end=‘\n’)
```

any number of positional arguments

specify separator character; defaults to space

specify end character; defaults to newline

```
print("mon", "tues”, sep=", ", end=", ")
print("wed", "thurs", sep=", end=", ")
print("fri", "sat", "sun", sep=", ")
print("cursor is now here")
```

Output:

mon, tues, wed, thurs, fri, sat, sun

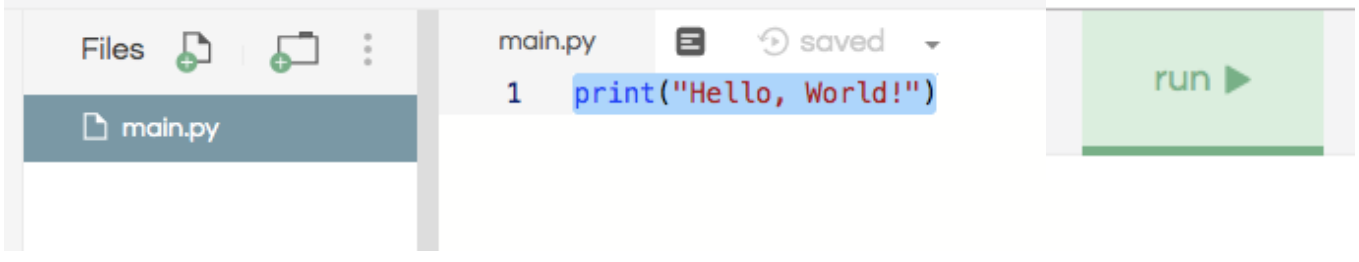
cursor is now here

Python Scripts

```
print("Hello, World!")
```

repl.it: Create a new repl and pick Python as the language, name the repl.

Type in the code in the file main.py. Click on run.




```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Hello, World!
>
```

A Syntax Example

```
In [4]:
    # set the midpoint
    midpoint = 5
    # make two empty lists
    lower = []
    upper = []
    # split the numbers into lower and upper
    for i in range(10):
        if (i < midpoint):
            lower.append(i)
        else:
            upper.append(i)
    print("lower:", lower)
    print("upper:", upper)
```

Comments Are Marked by #

End-of-Line Terminates a Statement

code blocks are denoted by *indentation (4 spaces)*

a *block* of code is a set of statements that should be treated as a unit.

indented code are always preceded by a colon on the previous line.

Dynamic Typing

Python is *dynamically typed*: variable names can point to objects of any type.

Unlike Java or C, there is no need to "declare" the variable.

```
In [1]: x = 1                # x is an integer
        x = 'hello'          # now x is a string
        x = [1, 2, 3]        # now x is a list
```

Basic Built-In Types

Type	Example	Description
`int`	`x = 1`	Integers (i.e., whole numbers)
`float`	`x = 1.0`	Floating-point numbers (i.e., real numbers)
`complex`	`x = 1 + 2j`	Complex numbers (i.e., numbers with a real and imaginary part)
`bool`	`x = True`	Boolean: True/False values
`str`	`x = 'abc'`	String: characters or text
`NoneType`	`x = None`	Special object indicating nulls

Built-In Function type()

```
In[7]: x = 4
       type(x)
Out [7]: int

In [8]: x = ‘hello’
        type(x)
Out [8]: str

In [9]: x = 3.14159
        type(x)
Out [9]: float
```

Integers

The most basic numerical type is the integer.

Any number without a decimal point is an integer.

```
In [1]: x = 1
        type(x)
Out [1]: int
```

Python integers are variable-precision, so you can do computations that would overflow in other languages.

```
In [2]: 2 ** 200
Out [2]:
1606938044258990275541962092341162602522202993782792835301376
```

Integers

By default, division upcasts integers to floating-point type:

```
In [3]: 5 / 2
Out [3]: 2.5
```

You can use the floor-division operator //:

```
In [4]: 5 // 2
Out [4]: 2
```

Floating Point

The floating-point type can store fractional numbers.

They can be defined either in standard decimal notation, or in exponential notation:

```
In [5]: x = 0.000005
        y = 5e-6
        print(x == y)
True
In [6]: x = 1400000.00
        y = 1.4e6
        print(x == y)
True
```

Floating Point Precision

One thing to be aware of with floating-point arithmetic is that its precision is limited, which can cause equality tests to be unstable.

```
In [8]: 0.1 + 0.2 == 0.3
Out [8]: False
```

```
In [9]: print(f'0.1 = {0.1:.17f}') # this is an f-string
        print(f'0.2 = {0.2:.17f}') # prints 17 characters
        print(f'0.3 = {0.3:.17f}') # after decimal point
0.1 = 0.10000000000000001
0.2 = 0.20000000000000001
0.3 = 0.29999999999999999
```

We will cover f-strings later in another lecture.

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O'reilly Media.

This book is completely free and can be downloaded online at O'reilly's site.