

# Introduction to Python

**Basic Datatypes**

**Integers, Floats, Booleans, NoneType**

# Topics

- 1) Python interpreters
- 2) Python Scripts
- 3) Printing with print()
- 4) Basic Built-In Number Types
  - a) Integers (division vs. floor division)
  - b) Floats
  - c) Booleans
  - d) NoneType

# 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). For windows, bundled with Anaconda is a set of programs. Open the “Anaconda Prompt” program. Then type “ipython”.

```
(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
```

# IPython Interpreter

Running Python snippets line by line using the IPython interpreter is useful if you're trying to experiment and learn Python syntax or explore how to solve a problem.

Every line is executed one at a time, allowing you to see output as you explore.

We will use the IPython interpreter to teach/learn Python syntax in these lecture slides.

# Python Scripts

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.

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

`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:

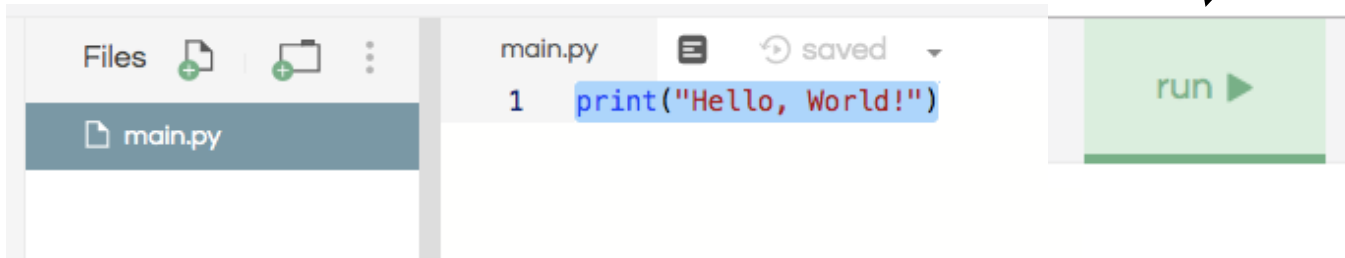
```
mon, tues, wed, thurs, fri, sat, sun
cursor is now here
```

# 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 [1]:
```

```
# initialize x
```

```
x = 5
```

Comments Are Marked by #

End-of-Line Terminates a Statement

```
if x < 10:
```

```
    x = x + 4
```

```
    print(x)
```

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.

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

```
print(x)
```

# 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 = 0.100000000000000001
```

```
0.2 = 0.200000000000000001
```

```
0.3 = 0.299999999999999999
```

**We will cover f-strings  
later in another  
lecture.**

# Casting

The `int()` and `float()` functions can be called to cast a value to a integer or float, respectively.

```
In[3]: x = int(1)    # x will be 1
        y = int(2.8) # y will be 2
        z = int('3') # z will be 3
        a = float(2.8) # a will be 2.8
        b = float("3") # b will be 3.0
        c = float('4.2') # c will be 4.2
```

# Boolean Type

The Boolean type is a simple type with two possible values: True and False.

Boolean values are case-sensitive: unlike some other languages, True and False must be capitalized!

Comparison operators return True or False values.

```
In[1]: x = False
```

```
      type(x)
```

```
Out[1]: bool
```

```
In [2]: result = (4 < 5)
      result
```

```
Out[2]: True
```

```
In[3]: 3 == 5
```

```
Out[3]: False
```

# NoneType

Python includes a special type, the `NoneType`, which has only a single possible value: `None`.

```
In [24]: type(None)
```

```
Out [24]: NoneType
```

`None` can is often used as the default value for parameters of methods. More on this later.

# Lab I

Download and install Anaconda on your home computer.

<https://www.anaconda.com/distribution/>

On your computer. Open the IPython interpreter. Experiment with some of the concepts we learn:

- 1) Print some strings. Experiment with the “sep” and “end” parameters.
- 2) Try division vs floor division with integers.
- 3) Initialize some floating point variables in both standard decimal notation and in exponential notation.
- 4) Experiment with the `type()` function.



# Lab 2

Create a new repl on [repl.it](https://repl.it).

Repeat some of the things you did from Lab 1 in the script `main.py`.

The IPython interpreter runs your code and display the outputs. In your script, you won't have this feature. Instead, use the `print()` function to print out the values of your variables.

# References

I) 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.