

Python Lesson 1 - Introduction

Welcome to your introduction to computer programming! We are going to be learning a computer programming language, which you can think of as learning a new foreign language, just as if you were going to learn a new speaking language (Bonjour!). Just as there are hundreds of actively spoken languages in the world, there are hundreds of widely used programming languages used actively today.

The best way to learn is to dive in!

1. Course Description

We are going to be using the Python Programming Language. In order to get started programming, we will uncover the fundamental programming concepts that are found across almost every programming language. Once you master them in Python, you can fully utilize them in Python or any other language you may choose to learn.

Note: On Learning to Program

I encourage you to go ahead and type in the examples provided. Then spend time playing and modifying them to your liking. If the program fails to do what you intended, simply restart, or at the very least copy & paste in the code and try to follow through the programs execution.

Let us begin!

2. Getting Setup

We will be using an online environment for writing our code today. Open your favorite web browser (e.g., Chrome, Safari, etc.) and go to <https://try.jupyter.org>. You should see a page that looks like this:

Files Running Clusters

Select items to perform actions on them. Upload New ▾ ↺

<input type="checkbox"/>	
<input type="checkbox"/>	communities
<input type="checkbox"/>	datasets
<input type="checkbox"/>	featured
<input type="checkbox"/>	Welcome Julia - Intro to Gadfly.ipynb
<input type="checkbox"/>	Welcome R - demo.ipynb
<input type="checkbox"/>	Welcome to Haskell.ipynb
<input type="checkbox"/>	Welcome to Python.ipynb
<input type="checkbox"/>	Welcome to Spark with Python.ipynb
<input type="checkbox"/>	Welcome to Spark with Scala.ipynb

To get started, click in the New button in the top right and select Python 3


Upload New ▾ ↺

- Text File
- Folder
- Terminal
- Notebooks
- Apache Toree - Scala
- Bash
- Haskell
- Julia 0.3.2
- Python 2
- Python 3**
- R
- Ruby 2.1.5

3. Data Types

Arithmetic

We will start with numbers and doing basic operations with these numbers. We can perform operations such as addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`).

Try typing these commands into the box next to `In []:` and then clicking the play button ().

```
In [ ]: |  
1. 2+5  
2. 7-5  
3. 5*4+2  
4. 8/2*4  
5. 2.7 *3
```

In math, we know there are different numbers, such as whole numbers, real numbers, integers, complex numbers, etc. The two most common in Python are floats (numbers with decimals) and int's (integers).

Let's see a quick example and introduce the print command.

MORE THAN YOU NEEDED TO KNOW RIGHT NOW: PRINT IS A FUNCTION THAT OUTPUTS THE CONTENTS FOLLOWING THE COMMAND OUT TO A CONSOLE.

```
1. print(int(5))  
2. print(float(5)) #Note, you will see the value 5.0,  
   because we are representing a decimal number.
```

Numbers themselves represent one of the fundamental ways to represent data in Python, but we can also represent data as text.

Strings

The second way to represent data as a string, which is a piece of data that represents text. A string is individually made up of a collection of one or more characters (`'A-Z'` , `'1-9'` , `'$'` , `'#'` , etc.).

Try typing in these commands next to `In []:`

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

^The string is the part that is represented in between the double quotes.

And congratulations! You just wrote your first real program, the notoriously famous HELLO WORLD program!

Let's try some more examples.

```
1. print('abcdefg')
   # Note we can use single or double quotes around a
   # string, but we cannot mix them.
2. print('123456')
   # Note: That the data type of the items between the
   # quotes is of a string. If we want it to be represented
   # as a number, we have to tell python to try to
   # cast(i.e. transform) it into another type.
3. print(int('123456'))
   # Same result as above, but this time an integer is
   # returned.
4. print(float('123456'))
   # We can do this once again with the float data type,
   # and we see even more clearly that 123456.0 is
   # returned.
```

Should I type in the pound (#) sign? Any text behind the # sign gets ignored by Python. This means you can write comments for yourself to remember what exactly you were trying to achieve. It's a great habit to write comments in your code.

Strings are one of the fundamental ways to represent data.

Variables

A variable is a container for data. It is a way to refer to some piece of data. Here are some simple examples.

```
thatPerson = "Mike"
```

Here we have a variable called thatPerson is assigned to the value Mike. We use the equals operator to assign what is on the left of the equation (thatPerson) to what is on the right of the equation ("Mike").

Note that how we name variables matters. 'thatPerson' is a different variable from 'ThatPerson' or 'ThAtPeRsOn'. This means we always have to be careful when typing in our variables. It also means as a rule of thumb, to not use the same phrase to name a variable more than one time.



```
In [2]: thatPerson = "Mike"
In [3]: ThatPerson = 'joe'
In [4]: ThatPerson
Out[4]: 'joe'
In [5]: thatPerson
Out[5]: 'Mike'
```

Our First Data Structure: List

A list is a versatile data structure in Python, in which we can store a sequence of data.

To create a list, we name it, just like we would a variable. We then list each element between brackets [and]. Each element in our list is then separated with a comma.

```
BestFriends = ['Willie', 'Mike', 'Tomoki']
```

Index	Value
0	Willie
1	Mike
2	Tomoki

We can access elements individually by doing the following.

```
In [7]: BestFriends[2]
Out[7]: 'Tomoki'
```

We can also add elements to our list, by appending them. When we append to a list, we update it by adding an element at the end. In the example below, we use the dot operator after the name of the list we want to modify. This then gives us access to Python's built-in functions that we can perform on that list.

```
BestFriends.append("Raoul")
```

Index	Value
BestFriends[0]	Willie
BestFriends[1]	Mike
BestFriends[2]	Tomoki
BestFriends[3]	Raoul

There are some other common list operations we may want to perform listed in this table. We are going to continue to use lists in future lessons, and see how powerful this simple data structure can be.

Python Code	Description	Result
<code>len(BestFriends)</code>	Get length of the list	4
<code>[1,2]+[3,4]</code>	Concatenate two lists	<code>[1,2,3,4]</code>
<code>'Mike' in BestFriends</code>	Test for membership in list	true

<pre>for x in BestFriends: print(x)</pre>	Iterate through all of the elements of the list.	Willie Mike Tomoki Raoul
---	--	-----------------------------------

Of particular interest above is the example with the ‘for’ keyword, which we have not seen. Type it in (note the tab in front of the ‘print’ command), see the results, and then we will explain further in the next section.

Some Hints on Using Jupyter

Now that we’ve warmed up with a few commands, I want to also give you some hints on the Jupyter notebook that we are using. These are short-cuts that will save you time as you are learning to program, so you can focus more on the fundamental details rather than always having to worry about the syntax (which is still very important!).

Want to edit the command you just typed? No problem, click on it, edit it, and click the Play button again.

Tired of typing long names like BestFriends? Try typing just Best and then press the Tab key. Watch it automatically complete what you had started to type.

Tired of having to use the mouse to click the Play button? You can press Shift+Enter to do the same thing.

Boolean Datatype

We are going to introduce one more data type, the Boolean. A Boolean is a value that can be either true or false. To a computer, this also means a value is either a 1 (true) or a 0 (false). Lets take a quick look at a sample below to see.

```
In [1]: IAMRightHanded = True

In [2]: IAMRightHanded
Out[2]: True

In [3]: print(int(IAMRightHanded))
1

In [4]: IAMLeftHanded = False
```

Booleans are important to know about, because a computer needs to be able to make binary decision (i.e. yes this expression is true, or no this expression false).

4. Program Constructs

Control Flow



Computers themselves are extremely obedient! From our previous examples, you can see how as soon as we enter a command, the computer will execute it right away without even thinking!

We may want to simulate some way within our programs to execute commands based on certain conditions. This is known as 'control flow', and we can visualize it as a graph.

The analogy to a control flow can be related to any decision you make during your life.

- If I need food, then go to the grocery store.
- If I have enough money, then buy an extra dessert.
- If I have days off of work, then buy a plane ticket to Hawaii.

Each of the statements above has the following pattern:

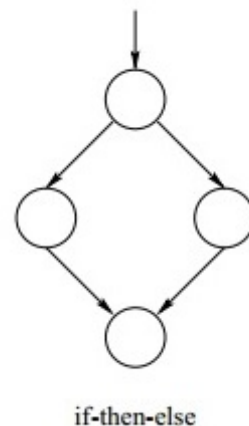
- the word 'if'
- some *condition* that can be evaluated to a Boolean (true or false value)
 - I need food
 - have enough money
 - have days off of work
- The series of events that happens if the *condition* is true.
 - go to the grocery store
 - buy an extra dessert
 - buy a plane ticket to Hawaii

We can use these values to then follow different paths in our control flow.

Conditional Statement

A condition statement is a statement that if a condition evaluates to true, then the block of code below it will execute.

Important Note: A block of code is the section of code that is indented after the colon. Code that is at the same level of indentation is in the same block. The indentation matters in Python, it is considered part of



the syntax to hit the tab key! The indentation also makes our code nice and easy to read, and it was a design choice by the Python programming language creators to enforce this.

Here is a first example of the if-statement. It will combine the concept of Booleans and control flow. Go ahead and type this in the IDLE editor, and be sure to hit enter twice after everything has been typed in.

```
joe_married = True
if joe_married:
    print('Status: Married')
else:
    print('Status: Single')
```

Another example of control flow introducing 'elif', which says, "if the first condition is not true, then check if this is true, else execute the following code by default". This pattern allows us to choose one of three blocks of code to execute based on some condition. In fact, if we want to, we can add as many 'elif' statements as we want (and the ending else condition is optional, only if we want to do something by default if none of the other conditions are satisfied).

```
joe_married = 2
if joe_married == 0:
    print('Status: Married')
elif joe_married == 1:
    print('Status: Single')
else:
    print("Status: It's complicated")
#Note that we have to use double quotes in this example to
wrap our string. Also note that we have changed joe_married
to be an integer value, because we want to evaluate
joe_married under more than 2 conditions (which with the
Boolean, we only can evaluate as true or false).
```

Instead of always using the '==' operator which tests if two values are equal, we can also use the following operators. Be careful with the '==', it does not work like the single '=' which means we are assigning a value. We want to test if two items are equal, as opposed to assign joe_married to a new value.

Operator	Description	Example	Result
<	Less than	1 < 2	True
>	Greater than	2 > 2	False
<=	Less than or equal to	2 <= 2	True
>=	Greater than or equal to	3 >= 2	True
==	Equal to	1 == 2	False

<code>!=</code> or <code><></code>	Not equal to	<code>1 != 2</code>	True
--	--------------	---------------------	------

MORE THAN YOU NEEDED TO KNOW RIGHT NOW: TESTING EQUALITY BETWEEN AN INT AND A STRING COULD BE PROBLEMATIC. TESTING EQUALITY BETWEEN A FLOAT AND A FLOAT COULD ALSO BE PROBLEMATIC! IN ALL CASES, IT HAS TO DO WITH HOW COMPUTERS REPRESENT DATA BEHIND THE SCENES.

Quick Review: Representing Data in Python

String – A collection of characters (More correctly stated: a list of characters)

Number – An int (...,-3,-2,-1,0,1,2,3,...) or a float (Decimal value such as 3.14)

Boolean – 1 (A true value) or 0 (A false value). We can alternatively use the keywords 'True' or 'False'

List- A collection of the types listed above.

Functions

We're starting to type a lot of text, and a long time ago programmers realized it is important to reuse code for their own sanity. A function in Python is similar to one we are use to in mathematics, such as $y(x) = x \cdot x$. We have a function 'y' that takes some parameter 'x', and then it returns a value based on the parameter x.

Lets go ahead and write that function using Python.

```
def square(x):
    return x*x
```

We define a new function with the Python languages 'def' keyword, give it a name 'square', and then any parameters that will be passed in. Finally we (optionally) return a value.

In our square example, when we use the actual function and pass in 2, then x is assigned the value '2' in that functions block of code, and we return $2 \cdot 2$ which returns a 4.

```
In [6]: def square(x):
        return x*x

In [7]: square(17)
Out[7]: 289

In [8]: square(3.14159)
Out[8]: 9.869587728099999
```

Functions can take more than one parameter if we like, here's another simple example of a function called 'add'.

```
In [11]: def add(x,y):
         return x + y

In [12]: add(8,23)
Out[12]: 31
```

It is actually okay for functions to not have any parameters at all (Note the parenthesis are just left empty).

```
In [13]: def sing():
          print("Happy Birthday to you")
          print("Happy Birthday to you")
          print("Happy Birthday dear Python")
          print("Happy Birthday to you")

In [14]: sing()

Happy Birthday to you
Happy Birthday to you
Happy Birthday dear Python
Happy Birthday to you
```

Let's write our `is_joe_married` function as well.

#Note we do not return anything in this function.

```
def is_joe_married(x):
    if x == 0:
        print('Status: Married')
    elif x == 1:
        print('Status: Single')
    else:
        print("Status: It's Complicated")
```

```
In [17]: def is_joe_married(x):
          if x == 0:
              print('Status: Married')
          elif x == 1:
              print('Status: Single')
          else:
              print("Status: It's Complicated")

In [18]: is_joe_married(0)

Status: Married

In [19]: is_joe_married(1)

Status: Single

In [20]: is_joe_married(2)

Status: It's Complicated

In [ ]: is_joe_married(-79)
```

Loops

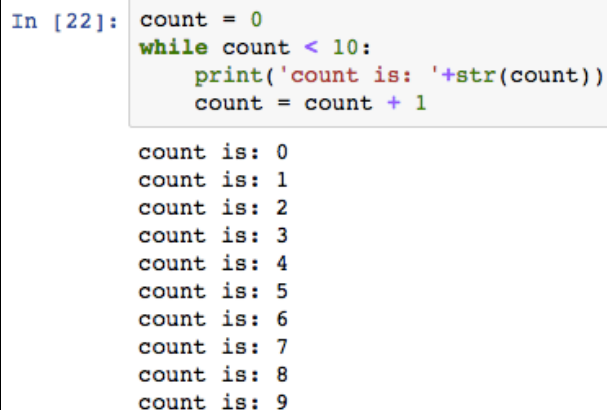
Loops are an important part of programming. They are a type control flow, in which we test a condition, and if it is true we continue to execute within the loop.

Note: With loops, often they can run forever if there is no condition. This is known as an *infinite loop*. Sometimes we want this behavior. For example, we always want to be able to use our e-mail client, and constantly have it returning new e-mails to us as people send them. But if we don't want this behavior, then we can press the Stop button (next to the Play button) to terminate the process.

While Loop

A while loop is a piece of code the executes while some condition is true at the top of the loop. Here's a few examples.

```
count = 0
while count < 10:
    print('count is: '+str(count))
    count = count + 1
```



The screenshot shows a Jupyter Notebook cell with the following code and output:

```
In [22]: count = 0
while count < 10:
    print('count is: '+str(count))
    count = count + 1
```

count is: 0
count is: 1
count is: 2
count is: 3
count is: 4
count is: 5
count is: 6
count is: 7
count is: 8
count is: 9

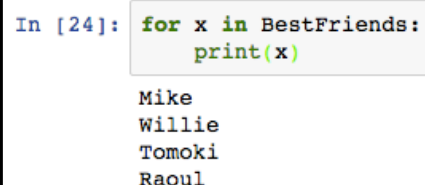
Note: In the previous code, the 'str' command was introduced. That converts a datatype into a string representation.

For Loop

A for loop is a piece of code that executes for a specified range. Semantically, we can achieve the same functionality as a while loop.

Iterating through a list:

```
for x in BestFriends:
    print(x)
```



The screenshot shows a Jupyter Notebook cell with the following code and output:

```
In [24]: for x in BestFriends:
        print(x)
```

Mike
Willie
Tomoki
Raoul

Iterating through a list again
over length of list:

```
for x in  
range(len(BestFriends)):  
    print(x)
```

```
In [25]: for x in range(len(BestFriends)):  
         print(x)  
0  
1  
2  
3
```

#Note the above seems unintuitive, but x is a variable that we're iterating through. So x is incrementing by one each time we find an element in the loop. Lets try that again with what we learned about accessing a list.

```
for x in range(len(BestFriends)):  
    print(BestFriends[x])
```

```
In [26]: for x in range(len(BestFriends)):  
         print(BestFriends[x])  
  
Mike  
Willie  
Tomoki  
Raoul
```

If that still seems a little
unintuitive, lets understand what
range is doing. The range
function gives you a list of
numbers that *range* from 0 up to
(but not including) the number
given. When we type in
len(BestFriends), that gives us a
number. So if we want a list of
numbers from 0 up to the length
of BestFriends we use
range(len(BestFriends)).
Finally, to get the item in the list
that corresponds with a number
in that range, we use the
number as an *index* into the list.

```
In [27]: range(5)  
Out[27]: range(0, 5)  
  
In [28]: len(BestFriends)  
Out[28]: 4  
  
In [29]: range(len(BestFriends))  
Out[29]: range(0, 4)  
  
In [30]: BestFriends[0]  
Out[30]: 'Mike'
```

5. Programming Challenge

Program 1: Guessing Game!

Goal:

- Randomly generate a number from 1 to 10.
- Take user input from user to guess the randomly generated number.
- When the user guesses the correct answer, output that they have finished.

Hint: To create a random number use the following Python function

```
import random # Bring in more Python functions
               # (This will be explained more in the next
               # section.)
answer = random.randint(1,10) #Includes both 1 and 10)
```

Hint: To get user input(stored as a string), use the 'input' command, but remember when using guess to cast it to an int if you do any comparisons.

```
guess = input('What is your guess?')
```

Program 1: Hints and suggestions

- You will need one loop that runs until the user guess matches the computer guess
- You will need two variables, one that keeps track of the answer and the guess.
- Use if statements to help the user make better guesses.
- Keep track of the number of guesses the user makes.
 - If they do it in less than 4 tell them they win!

The solution will be on the next page! Please do feel free to look after you've made at least one attempt.

Guessing Game Solution

```
import random
```

```
play = 1 # 1 is true
```

```
while(play==1):  
    print "Starting to play"  
    answer= random.randint(1,10)  
    guess=-1  
    guesses=0
```

```
# =====
```

```
# Playing the game
```

```
    while int(guess) != answer:  
        # (1) Let the user make a guess  
        guess=input("what is your guess")  
        # (2) Give the user feedback  
        if int(guess) < answer:  
            print("higher")  
        elif int(guess) > answer:  
            print ("lower")  
        # (3) They've made a guess, so increment guesses  
        guesses=guesses+1
```

```
# =====
```

```
# When you get your answer correct
```

```
    print('you got it'+str(guesses)+'!!')
```

```
# =====
```

```
# User feedback, after the game, telling them how well they did
```

```
    if guesses<5:  
        print('You are smart!')
```

```
    else:  
        print ("Luck was not on your side")
```

```
#-----
```

```
    # Ask the player if they'd like to play again
```

```
    # Ask the user if they want to play after they finish
```

```
    play = input("Would you like to play again? (1=yes or 0=no)")
```

```
import random

answer = random.randint(1,10)
guesses = 0
guess = -1

while guess != answer:
    guess = input('What is your guess?')
    if int(guess) < answer:
        print('Try a higher number')
    elif int(guess) > answer:
        print('Try a lower number')
    guesses = guesses + 1

print('You got it in'+str(guesses)+'!!')

if guesses<5:
    print('You are smart')
else:
    print('Luck was not on your side')
```

Going Further

Did you finish with extra time left, and want to do more before the next lesson?

- Try adding another loop and asking the user if they'd like to play again.
- Add another variable to keep track of the least number of guesses.
 - Perhaps increase the range to (1,100) to make this more interesting.
- Personalize the game, so that the first thing we do is ask for a name.
 - Add that name to a 'BestGuessingFriends' list.

6. Appendix

Low Level Details (More than you need to know now)

Python is an interpreted programming language. What this means, is that the computer reads Python code one line at a time and executes it.

