Python Quick Tutorial

Created by Harshit Agarwal and Andrew Yang

Printing and commenting is easy!

```
print "Hello world!" # This is a comment.
# This is another comment!
```

Declaring variables is also easy!

```
x = 42
y = -1
f = 1.21
s1 = "python"
s2 = '\"quote\"'
truuuuu = true
nuthin = None
```

In the above code, `x` and `y` are of type `int` (integer), `f` is of type `float` (floating point number), `s1` and `s2` are of type `str` (string), `truuuuu` is of type `bool` (boolean), and `nuthin` is of type `NoneType`.

Python supports arithmetic operations:

```
x + y # addition
x - y # subtraction
x * y # multiplication
x / y # division (floor division if both x and y are ints)
x % y # modulo
```

Python can also *simultaneously* do arithmetic and assign to a variable.

```
x += y # adds x and y and stores the sum to x
y -= x # subtracts x from y and stores the result to y
```

Similar operations exist for strings. "Adding" strings together is called concatenation.

```
x = 42
s = "My favorite number is "
s += str(x)
# Careful! You need to cast x from int to str.
```

Comparisons and If Statements

To compare variables, use these:

```
x == y # equality comparison
x != y # inequality comparison
x > y  # greater than comparison
x < y  # less than comparison
x >= y # greater than or equal to comparison
x <= y # less than or equal to comparison
```

Comparisons are important because they are used in if statements and loops, which help control the flow of the code.

```
x = 5
if (x > 0):
    # indentation is REQUIRED
    print "x is positive"
else:
    print "x is not positive :("
```

The output of the above code is `x is positive`.

Loops

For and while loops can execute a piece of code more than once:

```
myString = "snake"
for char in myString:
    print char

x = 3
while x > 0:
    print x
    x -= 1
print "Blast off!"
```

The output of the above code is:

```
s
n
a
k
```

```
e
3
2
1
Blast off!
```

Lists

Creating a list is easy:

```
myList = [1,2,5]
emptyList = []
```

In fact, you can iterate through a list using a for loop.

```
numbers = [8,6,7,5,3,0,9]
print "Here's Jenny's number:"
for number in numbers:
    print numbers
print "The first digit of Jenny's number is " + str(numbers[0])
```

The output:

```
Here's Jenny's number:
8
6
7
5
3
0
9
The first digit of Jenny's number is 8
```

Remember, lists are zero-indexed. That means that the first element of the list is located at index **0**.

Lists can be sliced so that only a segment of elements are included. The syntax goes as below:

```
myList[startIndex:endIndex]
# myList is a list, startIndex and endIndex are ints (both
```

The above code will slice myList from startIndex to endIndex, *excluding endIndex*. Assuming that `numbers` has already been declared as before:

```
print numbers[0]
print numbers[-1]   # print LAST element
for number in numbers[2:5]:
    print number
s = "python"
print s[2:]     # Strings can be sliced, too! And remember, the index arguments are optional!
```

The output is:

```
8
9
7
5
3
thon
```

Dictionaries

A dictionary is a set of key-value pairs.

```
eclectic = {false: "naw", 42: true, "dict": "JSON"}
letterNumbers = {"a": 1, "b": 2, "c": 3, "d": 4, "e": 5}
print letterNumbers["d"]
```

In the above code, for `letterNumbers`, `"a"` maps to `1`, `"b"` maps to `2`, etc. The output is `4`.

Functions and Objects

Declaring functions is easy.

```
def myFunc():
    print "This is my first function!"

def addTwo(num):
    print "The result is " + str(num + 2)
    return num + 2

def findOne(myList):
    for num in myList:
        if num == 1:
            print "One has been found"
            return
    print "One has not been found :("

myFunc()
print addTwo(5)
print findOne([4,2,1])
```

As you can see, functions can have parameters and can return values. If the function is finished with no return value, then the return value is `None`. The output is:

```
This is my first function!
The result is 7
7
One has been found
None
```

Objects are a collection of functions and variables. Once an object has been initialized, its constructor is called. Here's an example:

```
class myObject(object):
    def __init__(self, text):
        # This is the constructor.
        self.text = text
    def getText(self):
        return self.text
    def __str__(self):
        # Equivalent to Java's toString()
        return

o = myObject("Hello world")
print o
```

The object's variables (called instance variables) can be referenced through the `self` keyword. This is required as the first argument of all the object's functions (called methods). The above code outputs `Hello world` instead of something else because we overrode the `__str__` method.

Modules

You can augment your code with modules. Python has tons of these built-in modules and you can even install more modules using pip! (More info: http://pip.pypa.io/)

Here's how:

```
import math
import random
from string import digits
```

```
print "Cosine of 0 is: " + str(math.cos(0)) # Cosine of 0 is: 1
print random.random() # Prints a random float between 0 to 1, excluding 1.
print digits # 0123456789

# NOTE: The above line doesn't read print string.digits
# because we imported the variable INTO the global namespace rather than the module's namespace.
```

There are more modules than those listed. Go explore around!

Useful Functions

```
# raw_input(str) - Gets input from the user with an optional prompt.
userInput = raw_input("Say something, I'm giving up on you: ")

# len(str or list) - Returns the length of a string or list.
print len("python") # 6
print len([]) # 0

# range(int) - Returns a list of all ints from 0 to the argument, excluding the argument.
print range(3) # [0, 1, 2]

# range(int, int) - Returns a list of all ints from the first int to the second int, excluding th
print range(5,9) # [5, 6, 7, 8]

# range(int, int, int) - Acts like the above range function, but the third argument
# specifies the difference between each element.
print range(1,9,2)  # [1, 3, 5, 7]
print range(4,0,-1) # [4, 3, 2, 1]

# xrange works exactly the same as range, however xrange returns an object instead of a list.
# This is useful for iterating through a for loop.
for i in xrange(1000000):
    print i
# This prints the numbers 0 through 999999. Simply using range(1000000) would've used up a lot of

# sorted(list) - Returns the sorted list.
numbers = [8,6,7,5,3,0,9]
print sorted(numbers) # [0, 3, 5, 6, 7, 8, 9]
```