

# Intro to Python - Day 1

joseph.rejive

October 2018

Python is a powerful language that has many applications. It is the backbone of many major websites, including Google, Youtube, and Instagram. It's an easy language to pick up and can be used to create powerful programs, like language translation systems and computer vision models.

## 1 Your First Program

To start with, we'll create a "Hello World" program, which will display some text. Open a new file in your text editor and save it as `hello_world.py`.

---

```
print("Hello World")
print("How are you?")
```

---

If you're using Idle, select the run option at the top and click run module. On command prompt, navigate to the file you just created. Run the program by typing:

---

```
python hello_world.py
```

---

(Note on some machines, you may need to type either "py" or "python3" without the quotation marks in case the above command does not work).

Congrats! You've created your first program! As you may have guessed, the "print" statement displays text. Now let's see what else we can do with this statement...

## 2 Variables

In computer science, variables are merely names for something. Good variable names allow for clean, readable code. Here's how you can create variables in Python:

### 2.1 Numbers and Operations

First, we'll explore variables that are numbers and how to perform mathematical operations on them.

---

```
a = 5
b = 3
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a**b)
print(a%b)
```

---

In this example, we've created two variables, "a" and "b", that refer to two numbers, "5" and "3" respectively. Most of these operations are self explanatory, except for the last two. The double asterisk means power ( $5**3 = 5*5*5 = 125$ ), while the percent sign means modulus. The modulus operator returns the remainder of the division between two numbers ( $5\%3 = 2$ ).

## 2.2 Strings and Text

Strings are sequences of characters. Think of them as text.

---

```
string_1 = "800"
string_2 = '800'
integer = 800
print(type(string_1))
print(type(string_2))
print(type(integer))
```

---

“type” is a reserved keyword in Python that returns the datatype of the variable inputted into the function. Here, we can see that “800” and ‘800’ are instances of strings, while 800 is an integer. Mathematical operations do not work correctly on strings. To prove this, try running the following program.

---

```
a = "123"
print(a*3)
```

---

In the above program, instead of printing 369 (which is  $123 \times 3$ ), the output is “123123123”. When we multiplied “a” by 3, python took the string “123” and copied it 3 times. Watch out for these kinds of bugs when you’re coding.

## 2.3 The Input Function

Another reserved Python keyword is “input”. This function displays a prompt on the screen and accepts a user input. Lets explore how it works.

---

```
response = input("Enter Something: ")
print("You entered:", response)
print(type(response))
```

---

In this code snippet, we assign the “response” variable to the user input. If we take a look at the datatype, we can see that the input function always returns a string. Let’s take a look at some more examples.

---

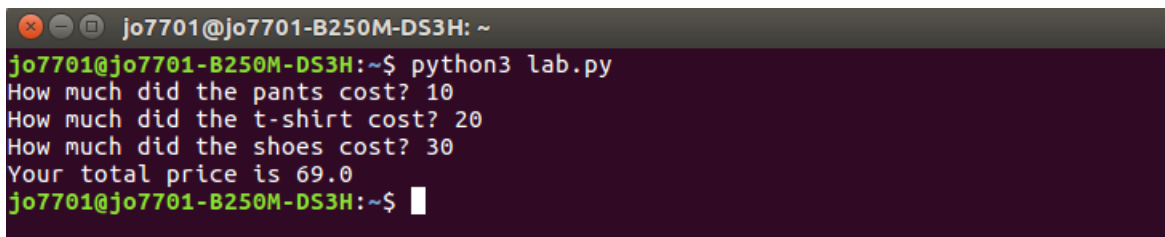
```
age = input("How old are you? ")
print(type(age))
age = int(age)
print(type(age))
print("In 10 years, you will be", age+10, "years old")
```

---

In the second line, we are ‘casting’ the age variable to an integer. As a result, age is no longer a string. We can verify this by checking its type.

## 2.4 Lab

For your first lab, you’ll be creating a program that will display the total cost of buying 3 articles of clothing. Your program should prompt the user to enter the price of each item. Given that the sales tax is 15% of the total price, your program should print out the total cost of the purchase. Below is an example.



```
jo7701@jo7701-B250M-DS3H: ~
jo7701@jo7701-B250M-DS3H:~$ python3 lab.py
How much did the pants cost? 10
How much did the t-shirt cost? 20
How much did the shoes cost? 30
Your total price is 69.0
jo7701@jo7701-B250M-DS3H:~$
```

## 3 Conditional Statements

Many times when you're programming, you'll want to execute a certain block of code only if certain criteria are met. Python allows you to do that through if-else statements.

---

```
name = input("What is your name? ")
if name == "Ryan":
    print("Welcome home Ryan!")
elif name == "Susan":
    print("Welcome home Susan!")
else:
    print("Please leave immediately.")
```

---

In the above example, we ask the user for an input and then check to see what name they gave us. If the name is Ryan, we say hello. The “elif” keyword stands for “else if”. If the name isn't Ryan and if it is Susan, then we also say hello. The “else” block contains code that will always run if the other two conditional statements are not met.

One unique feature of Python is the need to indent certain lines. Lines that follow a colon (":") will need to be indented. With that said, here's another example of conditional statements:

---

```
guess = input("Guess a number: ")
guess = int(guess)
answer = 15
if guess == answer:
    print("You guessed correctly!")
elif guess > answer:
    print("You guessed too high!")
else:
    print("You guessed too low!")
```

---

The “>”, “<”, and “==” operators are very commonly used in if statements. The double equals sign checks if the value on the left hand side equals the value on the right hand side. Make sure you don't confuse this with “=” because a single equals sign means you are assigning something to a variable (example: a = 15). Some other common symbols include “>=” (greater than or equal to), “<=” (less than or equal to), and “!=” (not equal to).

---

```
age = int(input("How old are you? "))
name = input("What is your name? ")

if age > 30 and name == "Ryan":
    print("Hello Ryan!")
elif age < 30 and name == "Tommy":
    print("Hello Tommy!")
else:
    print("You're an imposter!")
```

---

In the above code, we can see how we can check for multiple conditions in an if statement. When we're using the “and” keyword, the code in the if statement will only execute if both conditions are met. Another keyword we can use is “or”. This will execute code in the if statement if one or the other condition is met.

### 3.1 Mini Lab 1

Create a program which asks the user for a number and prints whether it is even or odd. (Hint 1: The input function returns a string) (Hint 2: What does the % operator do?)

## 4 Loops

Loops allow you to execute a block of code multiple times. There are two main types of loops: for loops and while loops.

### 4.1 While Loops

While loops execute a block of code repeatedly until a condition is met. Let's take a look at an example to figure out how they work.

---

```
counter = 0
while counter < 5:
    print("The value of the counter variable is:", counter)
    counter = counter + 1
print("The final value of counter is:", counter)
```

---

If we take a look at the output, we can see how counter went from 0 to 4. At every iteration, Python checked to see if the counter variable was still less than 5. Once counter became equal to 5, the code in the while loop did not execute. BE CAREFUL! A common error programmers make is forgetting to write `counter = counter + 1`. If you don't include this, counter will always stay at 0, which means counter will forever be less than 5. This results in an infinite loop. Try taking out `counter = counter + 1` and see what happens.

### 4.2 Mini Lab

Write a program that asks a user for a password. As long as the user enters the wrong password, your program should keep on asking the user for a password. Below is some code to get you started. (Hint: What does `!=` and `==` do?)

---

```
guess = input("What is the password? ")
password = "supersecret"
#write your code here
```

---

### 4.3 For Loops

A for loop works by repeating a code block a certain amount of times. Here's an example of how it works:

---

```
for counter in range(10):
    print("The value of the counter variable is:", counter)
```

---

As you can see, the code in the for loop executed 10 times. Another important thing to note is that counter started at 0 and ended at 9. This is because the "range" function always starts at 0 when you only give it one argument (arguments are just inputs to a function). Let's take a look at another example and give the range function 2 arguments.

---

```
for counter in range(10, 20):
    print("The value of the counter variable is:", counter)
```

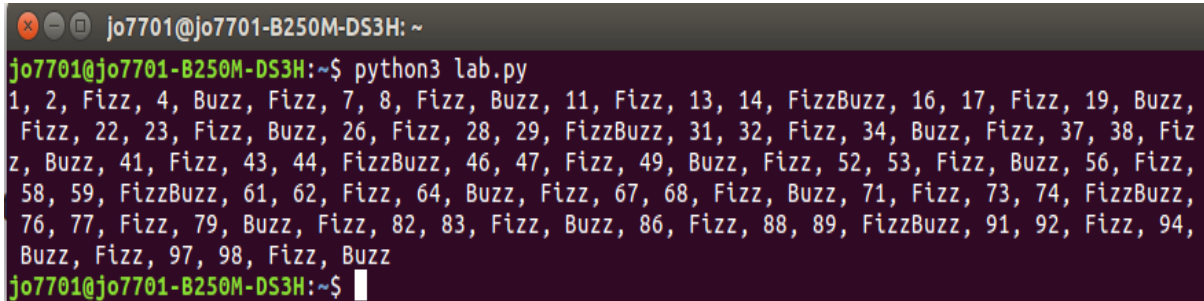
---

In this example, you can see how counter started at 10 and went all the way up to 19. When you pass in 2 arguments to the range function, you start counting at the first argument (in this case 10) and keep counting until one less than the second argument (in this case, one less than 20, which is 19).

### 4.4 Mini Lab

You'll be creating the FizzBuzz game. Write a program that counts from 1 to 100 (inclusive) and prints "Fizz" if the number is divisible by 3, "Buzz" if the number is divisible by 5, and "FizzBuzz" if the number

is divisible by both 3 and 5. If the number isn't divisible by 3 or 5, just print the number itself. Below is the expected output.



```
jo7701@jo7701-B250M-DS3H: ~  
jo7701@jo7701-B250M-DS3H:~$ python3 lab.py  
1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz, 16, 17, Fizz, 19, Buzz,  
Fizz, 22, 23, Fizz, Buzz, 26, Fizz, 28, 29, FizzBuzz, 31, 32, Fizz, 34, Buzz, Fizz, 37, 38, Fizz,  
Buzz, 41, Fizz, 43, 44, FizzBuzz, 46, 47, Fizz, 49, Buzz, Fizz, 52, 53, Fizz, Buzz, 56, Fizz,  
58, 59, FizzBuzz, 61, 62, Fizz, 64, Buzz, Fizz, 67, 68, Fizz, Buzz, 71, Fizz, 73, 74, FizzBuzz,  
76, 77, Fizz, 79, Buzz, Fizz, 82, 83, Fizz, Buzz, 86, Fizz, 88, 89, FizzBuzz, 91, 92, Fizz, 94,  
Buzz, Fizz, 97, 98, Fizz, Buzz  
jo7701@jo7701-B250M-DS3H:~$
```

## 5 Lists

The first datastructure we'll be talking about is lists. A list is a structure that can store objects. Here's an example of a list.

---

```
my_list = []  
print(my_list)  
my_list.append(10)  
print(my_list)  
my_list.append(9)  
print(my_list)
```

---

The empty brackets, “[]”, create an empty list. The append function adds a single element to the end of the list. One thing to note is the syntax we used: “my\_list.append()”. Before, the functions that we used did not have a variable name associated to it. The reason why we need to add “.append()” is because the append function belongs to the list class. Other functions we've used before, such as “input” and “range” did not belong to a certain class.

Here's another example of a list and how to access its elements.

---

```
my_list = [4, 6, 2, 9, 10]  
print(my_list[0])  
print(my_list[1])
```

---

In the example, we have a list of integers. To access elements in the list, we put the index of the element we want to access in brackets. One important concept to know is that lists are zero-indexed. This means the index of the first element is 0, not 1. Similarly, the index of the last element in this example is 4.

Now let's try to break this code:

---

```
my_list = [4, 6, 2, 9, 10]  
print(my_list[5])
```

---

When we run this program, we get an error: “list index out of range”. The reason this error happens is because the last element in my\_list is at index 4. When we try to access index 5 (which doesn't exist), Python throws an error. These kinds of errors are common so make sure to be careful when you're indexing lists.

---

```
my_list = [4, 6, 2, 9, 10]  
print(len(my_list))  
print(6 in my_list)  
print(145 in my_list)
```

---

To find the length of a list, we can use the built in “len” function. Another useful keyword is “in”. The statement “6 in my\_list” returns True or False, depending on whether 6 is present in the list.

## 5.1 Loops and Lists

A common way to access elements of a list is to use loops. Here’s an example:

---

```
my_list = [4, 6, 2, 9, 10]
for index in range(len(my_list)):
    print(my_list[index], "is the element at index", index)
```

---

This above code loops through every element in the list and prints it out. If you remember from the loops section, when the “range” function receives only 1 argument, the function counts from 0 to one less than the argument we provide. In this example, the argument we inputted to the range function was “len(my\_list)”. Since the length of the list is 5, the range function will count 0,1,2,3,4. It’s important to note that we don’t get a “list index out of range” error because the range function stops counting after 4.

---

```
my_list = [4, 3, 2, 9, 10]
for index in range(len(my_list)):
    if my_list[index] % 2 == 0:
        my_list[index] = 0
print(my_list)
```

---

In this example, we’re looking at each element in “my\_list” and setting it equal to zero if it is an even number.

## 5.2 For Each Loop

So far, we’ve always used the “range” function when we created for loops. Another kind of loop is the for each loop. Here’s an example of how it works.

---

```
list_of_names = ["Ryan", "Sarah", "Eric", "Carly"]
for name in list_of_names:
    print(name)
```

---

In this code, we’re looping through each element in “list\_of\_names”. We can access each element without using the “range” function. This is because the “name” variable in the for loop points to each element in the list. In addition to being very convenient, for each loops also increase the readability of your code. However, variable assignment will not work in a for each loop. To show this, take a look at the following code.

---

```
list_of_names = ["Ryan", "Sarah", "Eric", "Carly"]
for name in list_of_names:
    name = "Richard"
print(list_of_names)
```

---

This code will not change every name in the list to “Richard”. It doesn’t even change the list at all. If you wanted to change the names in the list, the correct way would be as follows:

---

```
list_of_names = ["Ryan", "Sarah", "Eric", "Carly"]
for index in range(len(list_of_names)):
    list_of_names[index] = "Richard"
print(list_of_names)
```

---

### 5.3 Lab

Create a program that finds the largest number in a list of integers. Below is some shell code to get you started.

---

```
a = [1,5,3,6,7,2,5,4,7,1,3,8,0,4,2]
#your code goes here
```

---

## 6 Practice Problems

Below are practice problems that increase in difficulty.

1. Create a program that generates the Hailstone series for a number. Here's how you do it: start with any number "n". If "n" is even, divide it by 2 and set that number as "n". If "n" is odd, multiply it by 3, add 1, and set that new number as "n". Continue this process until "n" equals 1.  
n = 5 gives the following sequence: 5, 16, 8, 4, 2, 1  
n = 14 gives the following sequence: 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
2. Write a python program that finds the common elements in two lists.  
If a = [3,4,5,6] and b = [5,6,7,8], your program should print a list that contains [5,6]
3. Given a list "my\_list", reverse it and print out the new list.  
If my\_list = [1,2,3,4,5], your program should print [5,4,3,2,1].
4. In the Fibonacci sequence, every number is the sum of the previous two numbers (except for the first two numbers, which are 1). Here is the first 10 digits of the sequence: 1,1,2,3,5,8,13,21,34,55. Create a program that accepts a number "n". Print the nth Fibonacci number. (Hint: store the sequence in a list)  
If n = 4, your program should print 3. If n = 8, your program should print 21. To check if your code works properly, you can Google the Fibonacci sequence.
5. Create a program that loops through the numbers 1-100 (inclusive) and prints all the prime numbers. A prime number is a number whose only factors are 1 and itself. Remember, 1 is not a prime number. (Hint: A for loop inside a for loop might be helpful)