# itpp-lesson2

January 12, 2018

## 1 CS190: Python

### 1.1 Lecture 2: Programming Fundamentals

Accompanying Slides

### 1.2 Homework

Homework 1 is due. You should have a file in your `turnin` folder called `[first_name]_[last_name]_[homework1].ipynb`. Grades/feedback will be released over the weekend.

### 1.3 Review

Last week we learned:

- There are hundreds of programming languages that have died, evolved, and created over the years
- How to write the "Hello World" program in 2 programming languages
- How to write in Markdown

Learning the fundamentals to programming will enable you to program in nearly any programming language! We've chosen Python for this course due to the wide variety of use cases that can be found online and in publications. Because of that, nearly any problem you run into while programming in Python can be solved by Googling.

### 1.4 Lesson Objectives

By the end of this lesson, you will learn:

- How to use **mathetical operators**
- How to use **variables** to store information
- How to use **comments** to document your code
- How to use **loops** to do repeated tasks
- Write **conditionals** to do actions based on a condition

---

## 1.5 Mathematical Operators

Performing math functions is useful in programming. You can think of a programming language as a glorified calculator. Here's how you can perform some common math operations in Python:

- Addition (5 + 5)

  ```
  5 + 5
  ```

- Subtraction (5 - 5)

  ```
  5 - 5
  ```

- Multiplication ($5 \cdot 5$)

  ```
  5 * 5
  ```

- Division ($\frac{5}{5}$)

  ```
  5 / 5
  ```

- Exponentiation ($5^5$)

  ```
  5 ** 5
  ```

  If precedence is needed, you can use parenthesis to dictate the order of the operations.

```
(5 * 6) + (5 - (3 + 2)) # Output is: 30 + 0 = 30
```

Reference on Python Mathematical Operators

### 1.5.1 Practice: Mathematical Operators

**Write the following in Python:** $5 \cdot 10$

```
In [1]: 5 * 10
```

```
Out[1]: 50
```

**Write the following in Python:** $5 \cdot (5^3 + 10)$

```
In [2]: 5 * (5**3 + 10)
```

```
Out[2]: 675
```

---

## 1.6 Variables

Variables allow use to store and access information. This is handy when we calculate a value but won't use it immediately after. Let's look at an example where we store the number 5 into a variable then output the sum between 5 and 6.

```
In [5]: x = 5
        print(x + 6)
```

```
11
```

**Practice: Store the value 10 into a variable y. Then print the output of $y^2 + 15$**

```
In [2]: y = 10
        print(y**2 + 15)
```

```
115
```

### 1.6.1 Data Types

You can think of a variable as a box. Boxes can be used to store variables and they can labeled so you know what's inside each box. Not all boxes are the same though.

In programming languages, there is a concept called data type. A data type is a designation at the hardware level on how information is represented. Some commons types are:

- `int`
- `double`
- `strings`
- `characters`

For most programming languages, you can't mix data types the same way as you can't compare apples and oranges. Even though both a fruit and have similar properties, they are not the same. Some languages forbid doing operations between different types.

Let's try adding a string with an int:

```
In [3]: # Adding a string with an int

        print("This is a string" + 5)
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-3-8c092ef518e4> in <module>()
          1 # Adding a string with an int
          2
    ----> 3 print("This is a string" + 5)
```

```
TypeError: must be str, not int
```

You can see the error produced when trying to add the two different data types. Most languages, including Python give us the ability to **type cast** values. What is typing casting? It is the operation that changes data in one type to another.

Let's rewrite the above example but this time type cast the `int 5` to a `string`.

```
In [4]: # Adding a string with a type casted int

        print("This is a string" + str(5))
```

```
This is a string5
```

In the example above, we type casted the value 5 which was of type `int` to the type `string` by using the `str()` function. There

Type casting reference

---

### 1.7 Comments

Comments are useful when writing code. It is always best to write code in the most descriptive way but in some cases it may not be so obvious. Each language has the capability of writing comments. Comments help you and other people know what's going on with your code.

Let's look at an example:

```
In [4]: # This code caculates the mean of a dataset
        # stored in the variable data

        data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

        sum = 0
        for d in data:
            sum += d
        print("Sum:", sum / len(data))
```

```
Sum: 5.5
```

Comments can also be used to temporarily disable some code. This is useful when you are trying to fix your program or want to try a different approach without having to delete your code.

```
In [11]: # This code will output the product of 1 to n

         n = 10
```

```
        sum = 0

        # This code outputs the sum 0
        # for i in range(n):
        #     print(i)
        #     sum += sum

        # This code ouputs a sum but it isn't the right sum
        # for i in range(n):
        #     print(i)
        #     sum += i

        for i in range(n + 1):
            print(i)
            sum += i

        print("The product of 1 to n is: ", sum)
```

```
0
1
2
3
4
5
6
7
8
9
10
The product of 1 to n is:  55
```

You can quickly comment/uncomment code by highlighting the text and press `ctrl + /`. Try commenting/uncommenting the code blocks in the above examples to see the differences.

---

### 1.8 Loops

Loops are a pretty cool tool to use in programming. Instead of explicitly writing the instruction to perform a task multiple times, you can write it once in a loop.

For example, if you wanted to print "Hello World" 5 times, you can do it in the following 2 ways:

```
In [1]: print("Hello World")
        print("Hello World")
        print("Hello World")
        print("Hello World")
        print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

```
In [11]: for x in range(5):
             print("Hello World")

Hello World
Hello World
Hello World
Hello World
Hello World
```

The first example gets the task done in 5 lines while the second example got it done in 2 lines. What if you want to print "Hello World" 100 or 1000 or even 9999 times? If you follow the first example, you would have to write the print instruction up to 9999 times. In the second example, we can just swap out 5 with the number of times we want to print it.

Try printing `"Hello World"` 10 times using the second example:

```
In [4]: for x in range(10):
            print("Hello World")

Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

So how does this magic happen?

```
for x in range(5):
    print("Hello World")
```

The first thing you do is write `for` which tells Python you are intending to write a for loop. Next you declare a variable, in this case it is `x`. This variable stores a counter which starts off at the value of 0. Next you write `in` which means as long as `x` is within the range of `range(5)` the `print("Hello World")` will execute. The `range(5)` tells Python that you want the counter `x` to go up to 5. Everytime `print("Hello World")` is executed, our variable increments.

Let's rewrite the loop in plain english:

```
x is 0, as long as x in the range of 5
    print "Hello World"
    x increments by 1
```

Let's combine everything we've learned so far. Write a program that calculates $\sum_{i=1}^{10} i$

```
In [22]: # declare a variable called sum which starts off at 0
         sum = 0

         for x in range(11):
             sum += x

         print("Sum of 1 through 10 is:", sum)
```

```
Sum of 1 through 10 is: 55
```

Why are we using `range(11)` when we just want the sum of 0 through 10?
**Computers start counting at 0** and **The range is exclusive for the upper-bound**.
Line 4 can be written mathematically as $(0, 11]$. 0 is included by 11 is not so it goes from 0 to 10.

---

Let's try writing a for loop now. Write a program that calculates $\sum_{i=1}^{15} i + 1$

```
In [ ]: sum = 0
        for x in range(16):
            sum = sum + x + 1
            # The above line can be rewritten as
            # sum += x + 1
            # With programming lanauges, there are tons of shortcuts
            # to do common actions

        print(sum)
```

There are other types of loops other than the `for` loop. Some common ones are `while` and `do-while`.
Loops Reference

---

## 1.9 Booleans

Booleans is a special type of data type. This type can only have 2 possibilities unlike the other data types that can hold infinitely many options.

Booleans can either be `True` or `False`. Booleans are useful when we want to know whether something is true or not.

Let's look at an example

```
In [2]: sleptLastNight = True

        if sleptLastNight:
            print("I'm well rested!")
        else:
            print("I should've slept last night")
```

```
I'm well rested!
```

Often times we'll want to now if a value is greater, less than, or equal to another value. Just like in math, programming languages have **comparison operators**.

Here are some frequently used ones:

- > Greater than
- >= Greater or equal to
- < Less than
- <= Less than or equal to
- == Equal
- != Not Equal

```
In [7]: print(5 > 0)
        print(5 >= 5)
        print(5 < 0)
        print(5 <= 6)
        print(5 == 6)
        print(5 != 5)
```

```
True
True
False
True
False
False
```

Comparison Operators Reference

## 1.10 Conditionals

Conditionals allow our program to make smart decisions. It would be cool if our program could perform an action based on a factor. Let's think about a program that prints `"It's Positive"` when a number is positive and prints `"It's Negative"` when a number is negative.

Writing out that program in English looks like:

```
We have a number x

if x is positive
```

```
    print "It's Positive"

if x is negative
```

Translating the English to Python gets you:

```
In [3]: x = 50

        if x > 0:
            print("It's Positive")
        if x < 0:
            print("It's Negative")

It's Positive
```

The above example has x as hard coded. Being hard coded means the value is fixed. What is you want a user to provide input? Let's modify the above example to allow for user input.

```
In [12]: x = int(input())

         if x > 0:
             print("It's Positive")
         if x < 0:
             print("It's Negative")

10
It's Positive
```

We modified line 1 changing it from `x = 50` to `x = int(input())`. For now, we can disregard what `int()` does and only focus on `input()`. Using `input()` allows you to store whatever the user types in and store it to a variable.

Conditionals Reference

---

### 1.11 Recap

This week we learned about:

- Math operators
- Variables
- Data Types
- Comments
- Loops
- Conditionals

These are most of the core building blocks that make up all programs. Mastery of these building blocks will enable you to build sophisicated programs!

---

## 1.12 Next Week

Next week, we will cover:

- User Input/Output
- How do we receive user input?
- Functions
- How can we build reusable programs?
- Modules
- How can we save time by using other people's code?

---

## 1.13 Homework 2

This week we learned a lot of material. Don't get overwhelmed! It only gets easier from here once you master these fundamentals. You can work independetly or with a partner on this assignment.

For the homework this week, do 1-3:

1. Create a new notebook naming it `[firstname]_[lastname]_[homework2]`. An example of this is `[john]_[pham]_[homework2]`.
2. Create a Markdown cell with the following information:

- A title
- Your name
- Your email

3. Write a program that calculates the tip needed for a meal.

- The tip will be 15% and the total price of the meal is $50
- There will also be a generosity value which will be added to your tip after it is calculated
- Your program should first output what the program does
- If the tip is greater than $15 then you will output 'wow that's a huge tip!'
- Lastly, your program will output: 'The total for the meal is [], [] tip at [] is [] and I tossed in an extra [] because of the great service'

### 1.13.1 Tips

First write out the program in plain English. Then convert the English program to Python. Remember to use variables in places where the value can change.