Python Basics and Jupyter Notebook

-- Dr. Meenakshi Nerolu

Python Basics

Python as a Calculator

Python can be used as a calculator to compute arithmetic operations like addition, subtraction, multiplication and division. Python can also be used for trigonometric calculations and statistical calculations.

Operations

Python can be used as a calculator to make simple arithmetic calculations.

In [1]:
```
# Addition: adds two operands
5+3
```

Out[1]: 8

In [2]:
```
# Subtraction: subtracts two operands
3-4
```

Out[2]: -1

In [3]:
```
# Multiplication : multiplies two operands
3*4
```

Out[3]: 12

In [4]:
```
# Division (Division (float): divides the first operand by the second)
15/5
```

Out[4]: 3.0

In [5]:
```
# Division
13/2
```

Out[5]: 6.5

In [6]:
```
# Division
3/4
```

Out[6]: 0.75

In [7]:
```
# Integer Division (Division (floor): divides the first operand by the second)
3//4
```

Out[7]: 0

In [8]:
```
# Integer Division
4//3
```

Out[8]: 1

In [9]:
```
#Modules
13%5
```

Out[9]: 3

In [10]:
```
#Modules - returns the remainder when the first operand is divided by the second
12%2
```

Out[10]: 0

In [11]:
```
#Modules
14%2
```

Out[11]: 0

In [12]:
```
#Modulus
-13%5
```

Out[12]: 2

In [13]:
```
# Power (Exponent): Returns first raised to power second
2**3
```

Out[13]: 8

In a Jupyter Notebook, `2^3` does not represent exponentiation (raising 2 to the power of 3). Instead, it performs a bitwise XOR (exclusive OR) operation between the numbers 2 and 3.

- Why 1?:
  - The binary representation of 2 is 10.
  - The binary representation of 3 is 11.
- XOR operation:
  - 10 (binary for 2)
  - 11 (binary for 3)

Result: 01 (binary for 1)

Precedence of Arithmetic Operators in Python

The precedence of Arithmetic Operators in Python is as follows:

1. P: Parenthesis
2. E: Exponential
3. M: Multiplication (Multiplication and Division have the same precedence)
4. D: Division
5. A: Addition (Addition and Subtarction have the same precedence)
6. S: Subtraction

For Example: how do you calculate $3 + 2 \times 2^2/2 - 1$ manually and what is your answer?

In [14]:
```
# Here is the Python output:
3+2*2**2/2-1

# Explanation: 3 + ((2 * (2 ** 2)) / 2) - 1
#              3 + ((2 * 4) / 2) - 1
#              3 + (8 / 2) - 1
#              3 + 4 - 1
#              7 - 1
#              6.0
```

Out[14]: 6.0

In [15]:
```
#_ will be helpfull to use previous output.
_+5
# Here the output is 11 because _ uses the previous output 6 and 5 is added to it.
```

Out[15]:
```
11.0
```

Variables

What are Variables in Python?

A variable in programming is used to store information that you want to re-use in your code.

Examples of things that you may wish to save in your code include:

- numeric values
- filenames
- paths
- or even larger datasets such as a remote sensing image or a terrain model

In **Python**, variables can be created without explicitly defining the type of data that it will hold (e.g. integer, text string). Variables are assigned in Python using the `=` equals sign also called *assignment operator*. You can create a variable in **Python** using the following syntax:

```
variable_name=value
```

The statement:

In [16]:
```
a=2
```

assigns the integer 2 to the variable a.

Here is an example of assigning a text string value to the same variable named `a`.

In [17]:
```
a='a word'
```

Notice that strings (character values) use single/double quotes `''` or `""` to indicate a text string value.

In [18]:
```
#Try it out
my_variable=5
my_variable=5
```

Variable names in Python must conform to the following rules:

- variable names must start with a letter
- variable names can only contain letters, numbers, and the underscore character_

- variable names cannot contain spaces
- variable names cannot include punctuation
- variable names are not closed in quotes or brackets

When naming a variable, you want to keep the name short but specific enough that someone reading your code can understand what it contains. It is good practice to use underscores (e.g. `boulder_precip_in`) to create multi-word variable names that provide specifics regarding the variable's content. The underscore makes the variable name easier to read.

Variables Are Available In Your Coding Environment Once Defined

A key characteristic of variables is that once you create a variable in your coding environment (that is to say you run the actual line of code that defines the variable), it is available throughout your code. So for example, if you create a variable at the top of a Jupyter Notebook, the value associated with that variable will remain the same and can be reused in cells lower down in the notebook.

The following code lines show invalid variable names:

In [19]:
```
a constant = 4
#Note: space is not allowed
```

```
  Cell In[19], line 1
    a constant = 4
      ^
SyntaxError: invalid syntax
```

In [ ]:
```
123var =77
# variable name should start with letter
```

In [ ]:
```
'Nerolu' =last_name
#variable name should appear first
```

The following code lines show valid variable names:

In [ ]:
```
constant = 4

new_variable = 'var'

my2rules = ['rule1','rule2']

SQUARES = 4
```

In [ ]:
```
SQUARES
```

String Operations

Strings are sequences of letters, numbers, punctuation, and spaces. Strings are defined at the Python by enclosing letters, numbers, punctuation, and spaces in single quotes `' '` or double quotes `" "`. For example,

In [ ]:
```
word = 'Solution'
another_word = "another solution"
third_word = '3rd solution'
```

In Python, string operations include concatenation (combining strings), logical comparisons (comparing strings) and indexing (pulling specific characters out of strings).

String Concatenation

Strings can be concatenated or combined using the `+` operator.

In [ ]:
```
first_name = "Meenakshi"
second_name = "Nerolu"
Full_name = first_name +second_name
```

In [ ]:
```
Full_name
```

To include spaces in the concatenated string, add a string which just contains one space `" "` in between each string you combine.

In [ ]:
```
Full_name_new = first_name +" "+second_name
```

In [ ]:
```
Full_name_new
```

String Comparison

Strings can be compared using the comparison operator; the double equals sign `==`. Note the comparison operator (double equals `==`) is not the same as the assignment operator, a single equals sign `=`.

In [ ]:
```
name1 = 'Sijan'
name2 = 'Sijan'
name1 == name2
```

In [ ]:
```
name1 = 'Bria'
name2 = 'Alan'
name1 == name2
```

Capital letters and lower case letters are different characters in Python. A string with the same letters, but different capitalization are not equivalent.

In [ ]:
```
name1 = 'Sijan'
name2 = 'sijan'
name1 == name2
```

String Methods

Python String capitalize()

The `capitalize()` method converts the first character of a string to an uppercase letter and all other alphabets to lowercase.

In [ ]:
```
sentence = 'i love PYTHON'

# converts first character to uppercase and others to lowercase
capitalized_string = sentence.capitalize()

capitalized_string

# Output: 'I love python'
```

The `capitalize()` method returns a new string and doesn't modify the original string.

In [ ]:
```
first_name = 'meenakshi'
```

In [ ]:
```
# upper() is used to convert  the string from lowercase letter to Uppercase letter
first_name.upper()
```

In [ ]:
```
last_name = "NEROLU"
```

In [ ]:
```
# lower() is used to convert the stringfrom uppercase letter to lowercase letter
last_name.lower()
```

In [ ]:
```
full_name = 'Meenakshi   Nerolu'
```

In [ ]:
```
# Split
full_name.split("  ")
```

Print Statements

One built-in function in Python is `print()`. The value or expression inside of the parenthesis of a `print()` function "prints" out to the REPL when the `print()` function is called. An example using the `print()` function is below:

In [ ]:
```
name = "Zoe"
print("Your name is: ")
print(name)
```

Remember that strings must be enclosed by quotation marks. The following command produces an error.

In [ ]:
```
print(Zoe)
```

Strings can be concatenated (combined) inside of a `print()` statement.

In [ ]:
```
name = "Noor"
print("Your name is: ", name)
```

In [ ]:
```
name = "Noor"
print("Your name is: "+ name)
```

The `print()` function also prints out individual expressions one after another with a space in between when the expressions are placed inside the `print()` function and separated by a comma.

In [ ]:
```
print("Name:","Elise","Age", 2+7)
```

Notice that the word `print` does not show up the output. Instead, you simply see the result, without the parentheses or quotations for the text string.
You have now used your first Python function - `print()` ! Functions in Python are commands that can take inputs that are used to produce output. You will learn more about functions later in the following week, and you will use the `print` function a lot, as it can be very handy for viewing results and for communicating the status of your code.

More about operators

Relational Operators in Python

Often in Python, you need to compare two values against each other. To do this, you can check a statement, such as `3 < 4`, and get returned one of two values from Python: `True` or `False`. These are called boolean values and can be very powerful in scripting workflows. A boolean is a value that is either 1 (True), or 0 (False). Like `strings` or `integers`, booleans are their own data type.

In Python, there are many relational operations that can be used, including operators for:

- equal (`==`)
- not equal (`!=`)
- greater than (`>`)
- greater than or equal to (`>=`)
- less than (`<`)
- less than or equal (`<=`)

Review the cells below to see what these operations return in different circumstances.

In [ ]:
```
# Is the value 3 less then 4?
3 < 4
```

In [ ]:
```
# Is the value 3 greater than 4?
3 > 4
```

In [ ]:
```
# Does 3 equal 3?
3 == 3
```

In [ ]:
```
# Does 3 equal 4?
3 == 4
```

In [ ]:
```
# Does 3 NOT equal 4?
3 != 4
```

In [ ]:
```
# Is 3 less than or equal to 4?
3 <= 4
```

In [ ]:
```
# Is 3 less than or equal to 3?
3 <= 3
```

In [ ]:
```
# Is 3 greater than or equal to 4?
3 >= 4
```

In [ ]:
```

```