Python Basics

Objectives:

- Input and Output
- Basic Data Types
- More on Strings

1. Input & Output

1.1 Get User Input

You can ask user for some input using Python built-in `input()` function.

- When `input()` function is called, the program will stop and wait for user to key in some data.
- It is optional to add prompt as function parameter.
- It always returns a string value.

<u>Exercise:</u>

Ask user for his age and assign it to variable `age` .

In [1]:
```
age = input('How old are you?  ')
print(age)
```

```
How old are you?  18
18
```

What is the output from `type(x)` command?

In [2]:
```
type(age)
```

Out[2]:
```
str
```

Learn more about the `input()` function from help:

In [3]:
```
help(input)
```

```
Help on method raw_input in module ipykernel.kernelbase:

raw_input(prompt='') method of ipykernel.ipkernel.IPythonKernel instance
    Forward raw_input to frontends

    Raises
    ------
    StdinNotImplentedError if active frontend doesn't support stdin.
```

1.2 Print Out

Python provides a built-in function `print()` to display data to the console or a file object.

```
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Understand more about `print()` function:

- It can accept multiple values, and uses `sep` value to separate them in output. By default, the `sep` value is a space( ).
- It automatically appends an `end` string at the end. By default, the `end` value is a new line (`\n`).
- It outputs data to a file object defined in `file` , which has a default value `sys.stdout` (console).

<u>Exercise:</u>

In [4]:
```
print('hello', 'world')
print('have', 'a', 'good', 'day', sep='-')
```

```
hello world
have-a-good-day
```

2. Basic Data Types

In programming, a **data type** defines the type of a data and its expected behavior.

- Variables are commonly associated with a particular data type.

2.1 Overview of Data Types

Python support many data types, which allow us to implement solutions quickly using Python.

- Common data types: boolean, integer, float, string
- Collection data types: List, Tuple, Dictionary and Set

Question: How to check the data type of a variable?

2.2 Booleans

A Boolean variable can represent either `True` or `False` .

- Boolean values are commonly associated with comparison operators, which will be covered in another session.

Exercise:

In [5]:
```
x = (1==1)
y = (1=='1')
print(x, y)
type(x)
```

```
True False
```

Out[5]:
```
bool
```

2.3 Numbers

Numbers in Python are classified into **int**, **float**, and **complex**.

- The **int** type can contain integer values
- The **float** type can contain decimal values
- The **complex** type can contain imaginery values. Add a “j” or “J” after a number to make it imaginary or complex.

Exercise: Run following statements line by line. (Press `CTRL + ENTER` after typing each line)

In [6]:
```
type(123)
type(1.23)
type(1 + 2j)
```

Out[6]:
```
complex
```

2.4 Strings

String is a sequence of one or more characters enclosed within any of following quotes.

- single-quotes `' '`
- double-quotes `" "`
- Triple single-quotes `''' '''`

Python supports multi-line strings using a triple single-quotation mark at the start and one at the end.

Exercise:

In [7]:

```
x = 'Hello'
y = "World"
z = """Hello
agian"""
print(x, y)
print(z)
```

```
Hello World
Hello
agian
```

Exercise: For these examples, it uses combination of single-quote (') and double-quote (") instead of escape character.

In [8]:

```
print("I'd work hard")
print('"Go home", he said')
```

```
I'd work hard
"Go home", he said
```

**Escape Character **

Similiar to other programming langugae, Python uses \ (backslash) as escape character in strings.

Here are some common escape characters that are represented using backslash notation.

Symbol	Character
\"	Double-quote (")
\'	Single-quote (')
\n	Line feed (LF) or new line
\t	Horizontal Tab (TAB)
\\	Backslash ()

Exercise: Escape characters in Strings

In [9]:

```
print('hello\tworld')
print('Folder:\n\tc:\\Windows\\System')
```

```
hello	world
Folder:
	c:\Windows\System
```

3. String Interpolation

Instead of using + operator to concatenate strings, Python provides several other ways to perform string interpolations.

3.1 Using f-strings

Python 3.6 adds a new string interpolation method using prefix `f` .

In [15]:

```
x = 10
y = 'Ten'

z = f'{x} is {y}'
print(z)
```

```
10 is Ten
```

3.2 Using str.format()

The string object has a format() function which performs simple positional formatting.

In [17]:

```
name = 'world'
greeting = 'Hello, {}'.format(name)
print(greeting)
```

```
Hello, world
```

4. Type Conversion

4.1 Type Casting

There may be times when you want to specify a type on to a variable. This can be done with casting.

Casting can be done using constructor functions:

- `int()` - constructs an integer number
- `float()` - constructs a float number
- `str()` - constructs a string

<u>Exercise:</u> Convert values `123` to string, `"123"` to integer, and `"1.23"` to float.

In [11]:

```
str(123)
int("123")
float("1.23")
```

Out[11]: `1.23`

4.2 Dynamically and Strongly Typed

Dynamically Typed language: Type of a variable is only evaluated during run time, e.g. during variable creation.

Strongly Typed language: Variables of different type will not be automatically converted before operation.

Exercise: Correct the mistake in following code.

```
x = 'abc' + 99
```

In [ ]:

```
1
```