# Python
for everybody

Session 3

4. String

4. String

4.1. String vs List

String vs List

A string is a sequence of characters.

```python
fruit = "banana"
```

Strings are **not** lists, but they share many features:

```python
letter = fruit[1]
print(letter)

print('b' in fruit)

sliced_word = fruit[0:3]

for char in fruit:
    print(char)
```

String vs List

Strings are **immutable**, which means you can’t change an existing string :

```python
fruit = "banana"
fruit[0] = 'v' #Error
```

The best you can do is create a **new string** that is a variation on the original:

```python
fruit = "banana"
new_fruit = 'v' + fruit[1:]
print(new_fruit)
```

4. String

4.2. String Methods

String methods

```python
fruit = "banana"

index = fruit.find('b')       # Returns index of 'b'which is 0

characters = list(fruit)      # ['b', 'a', 'n', 'a', 'n', 'a']

full_name = "Mohamed Ali Mahmoud"
names = full_name.split(" ")  # ['Mohamed', 'Ali', 'Mahmoud']

words = ['look', 'for', 'a', 'job']
sentence = " ".join(words)    # 'look for a job'
```

5. Functions in Python

5. Functions in Python

5.1. Definition

Definition

A function is a named sequence of statements that performs a specific task.

In other words, it allows you to group several instructions under one name and execute (or “call”) them whenever needed.

5. Functions in Python

5.2. Built-in functions

Built-in functions

We have already used several functions such as print(), len(), etc.
These are called built-in functions, meaning they are predefined by Python.
Examples:

```python
print("Hello, World!")          # Output: Hello, World!

length = len("AppsClub")        # length = 8

sys.exit()                      # Exits the program

value = random.random()         # Returns a random float between 0 and 1
```

5. Functions in Python

5.3. Why do we use functions?

Why do we use functions?

We create our own functions for several reasons:

- They make the code **easier to read** by naming what the block does.
- They **eliminate repetitive** code blocks
- They help us **debug faster** and more efficiently
- They allow us to **reuse** code across different programs

5. Functions in Python

5.4. Defining your own functions

Defining your own functions

Syntax:

```python
def function_name(parameter1, parameter2, ...):
    # Function body (instructions)
```

To call a function:

```python
function_name(argument1, argument2, ...)
```

Defining your own functions

Example 1:

```python
def welcome_user(name):
    print("Welcome, Mr.", name)

welcome_user("Ahmed")
```

Example 2:

```python
def sum_numbers(x, y, z):
    print(x + y + z)

sum_numbers(6, 2, 3)
```

Notes:

- Funtion with default parameters

```python
def welcome_user(name="X"):
    print("Welcome, Mr.", name)

welcome_user("Ahmed")    # Welcome, Mr. Ahmed
welcome_user()           # Welcome, Mr. X
```

- Function with no arguments

```python
import random
def random_message():
    value = random.random()
    if value > 0.5:
        print("You win!")
    else:
        print("You lose!")

random_message()
```

Defining your own functions

Return statement

In previous examples, the functions printed results.

But functions can also return a value, which can be stored and reused.

Example 1:

```python
def sum_numbers(x, y, z):
    return x + y + z

S = sum_numbers(10, 20, 30)
print(S)    # 60
```

Example 2:

```python
def split_link(link):
    return link.split('/')

splitted_link = split_link("www.school.com/Bac/Math/Exercices")
# ['www.school.com', 'Bac', 'Math', 'Exercices']
```

5. Functions in Python

5.5. Variable scope in Python

Variable scope in Python

Example:

```python
a = 10

def change_value():
    a = 20

change_value()
print(a)
```

What do you expect the value of a to be? *It is still 10.*

Why ? Because assigning a value to a inside the function makes Python treat it as a new local variable inside the function's scope. It does not modify the variable a from the global scope.

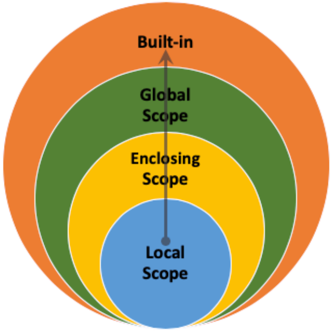
Variable scope in Python

Solution: If you want to modify the global variable from inside the function, you must explicitly tell Python by using the keyword **global**.

```python
a = 10

def change_value():
    global a
    a = 20

change_value()
print(a)    # Now prints 20
```


Built-in
Global Scope
Enclosing Scope
Local Scope