What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Good to know

The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular. In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

In [1]:
```
#Example
print("Hello, World!")
```

```
Hello, World!
```

Variables

Variables are containers for storing data values.

Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

In [2]:
```
#Example
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

note:

Variables do not need to be declared with any particular type, and can even change type after they have been set.

In [5]:
```
x = 5 # x is of type int
x = 'DENACO' # x is now of type str
print(x)
```

```
DENACO
```

Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

In [6]:
```
print(10 + 5)
```

```
15
```

In [14]:
```
# Basic Arithmetic:
print( 2 * 3 )       # Basic Arithmetic: +, -, /, *
print( 2**3 )        # Basic Arithmetic: +, -, /, * Exponentiation
print( 10 % 3 )      # Modulus Op. : returns remainder of 10/3
print( 1 + 2 * 3 )   # order of operations
print(10 / 3.0)      # int's and doubles : Division
print(10 // 3)       #Floor division
```

```
6
8
1
7
3.3333333333333335
3
```

Lists

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are `Tuple`, `Set`, and `Dictionary`, all with different qualities and usage.

Lists are created using square brackets:

In [17]:
```
mylist = ["apple", "banana", "cherry", "apple", "cherry"]
print(mylist)
```

```
['apple', 'banana', 'cherry', 'apple', 'cherry']
```

List items are:

1. ordered:

- means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list.

1. changeable:

- meaning that we can change, add, and remove items in a list after it has been created

1. allow duplicate values.

In [19]:
```
friends = ["Albert"]
friends.append("Oscar")
friends.append("Angela")
friends.insert(0, "Kevin")
print(friends)
```

```
['Kevin', 'Albert', 'Oscar', 'Angela']
```

Conditions and If statements:

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "`if statements`" and `loops`.

An "`if statement`" is written by using the if keyword.

In [22]:
```
# Example
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
b is greater than a
```

Note:

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

In [25]:
```
is_male = False

if is_male:
    print("You are a male")
else:
    print("You are a female")
```

```
you are a female
```

Python Loops

Python has two loop commands:

- while loops
- for loops

The while Loop

With the `while loop` we can execute a set of statements as long as a condition is true.

In [27]:
```
#Example Print i as long as i is less than 6:
# Note: remember to increment i, or else the loop will continue forever.
i = 1
while i < 6:
  print(i)
  print("Loop")
  i += 1
```

```
1
Loop
2
Loop
3
Loop
4
Loop
5
Loop
```

The break Statement

With the `break` statement we can stop the loop even if the while condition is true:

In [28]:
```
#Example: Exit the loop when i is 3:
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

```
1
2
3
```

In [30]:
```
is_student = True
is_smart = False

if is_student and is_smart:
        print("You are a student")
elif is_student and not(is_smart):
        print("You are not a smart student")
else:
        print("You are not a student and not smart")
```

```
You are not a smart student
```

For Loops

A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the `for` keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the `for` loop we can execute a set of statements, once for each item in a list, tuple, set etc.

In [35]:
```
#Print each lucky number in a lucky numbers list list:

lucky_nums = [4, 8, 15, 16, 23, 42]
for lucky_num in lucky_nums:
    print(lucky_num)
```

```
4
8
15
16
23
42
```

In [32]:
```
#Loop through the letters in the word "banana":

for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

In [33]:
```
#Exit the loop when x is "banana":
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

In [34]:
```
for index in range(5):
    print(index)
```

```
0
1
2
3
4
```

Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

In [36]:
```
#Example
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

In [39]:
```
def my_function(fname):
  print(fname + " is a Student at the university of Nairobi")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Emil is a Student at the university of Nairobi
Tobias is a Student at the university of Nairobi
Linus is a Student at the university of Nairobi
```

In [40]:
```
my_function("Albert")
```

```
Albert is a Student at the university of Nairobi
```

In [ ]:
```

```

In [46]:
```
# Let's define a mathematic formula using a function, We keep multiplying the base num with pow_num (for loop)
def raise_to_power(base_num, pow_num):
    results = 1
    for index in range(pow_num):
        results *= base_num
    return results
```

In [49]:
```
raise_to_power(10, 2)
```

Out[49]:
```
100
```