Introduction to Python section 1

By
Rhma Yasser

Grades:

- Attendence (5)
- Tasks & Quizes (10)
- Project – Practical (15)

Note: Don’t enter the class after 15 minutes.

Table of content

Resources:

1. Python crash course book

https://khwarizmi.org/wp-content/uploads/2021/04/Eric_Matthes_Python_Crash_Course_A_Hands.pdf

2. Python documentation

https://docs.python.org/3/tutorial/index.html

3. Geeks for Geeks

https://www.geeksforgeeks.org/python-programming-language-tutorial/

4. W3 schools

https://www.w3schools.com/python/

Other:

YouTube playlists

https://www.youtube.com/watch?v=h3VCQjyaLws&list=PLuXY3ddo_8nzrO74UeZQVZOb5-wIS6krJ

https://www.youtube.com/watch?v=mvZHDpCHphk&list=PLDoPjvoNmBAyE_gei5d18qkfIe-Z8mocs

Coursera

https://www.coursera.org/specializations/python-3-programming

01

Introduction to Python

Jupyter Installation:

https://www.youtube.com/watch?v=WUeBzT43JyY
https://www.youtube.com/watch?v=8YGPfGDYAgI

What is python?

1. Python is a high-level, interpreted programming language known for its simplicity and readability.

2. Python is an easy language to learn because of its simple syntax.

3. It is widely used in various domains such as web development, data science, artificial intelligence, machine learning, and more

Why is python?

1. **Community and Documentation:** Python has a large and active community of developers who contribute to its growth and development.

2. **Extensive Libraries:** Python has a vast ecosystem of libraries and frameworks for various purposes, allowing developers to leverage existing tools for their projects.

3. **Versatility:** Python can be used for a wide range of applications, from web development to scientific computing and machine learning.

01

Input – Output

Input - Output

How the input() Function Works

```
message = input(“Enter Hello World")
```

How the print() Function Works

```
print(“Hello World")
```

02

Comments

Comments

In Python, the hash mark (#) indicates a comment. Anything following a hash mark in your code is ignored by the Python interpreter.

```
#Say hello to everyone.
print("Hello Python people!")
```

03

Variables & Data Types

Naming and Using Variables

1. Variable names can contain only letters, numbers, and underscores. They can start with a letter or an underscore, but not with a number. For instance, you can call a variable ***message_1*** but not ***1_message***.

2. Spaces are not allowed in variable names, but underscores can be used to separate words in variable names. For example, ***greeting_message*** works, but ***greeting message*** will cause errors.

3. Avoid using Python keywords and function names as variable names.

4. Variable names should be short but descriptive. For example, ***name*** is better than **n**, ***student_name*** is better than ***s_n***, and ***name_length*** is better than ***length_of_persons_name***.

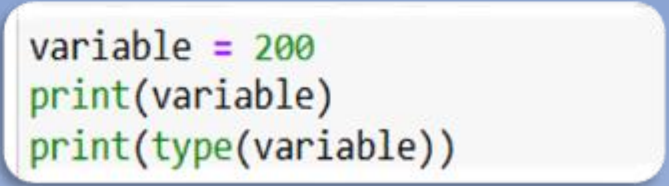
Variable types

1. Integer (int)
2. Float (float)
3. String (str)
4. Boolean (bool)
5. Complex (complex)
6. [...]

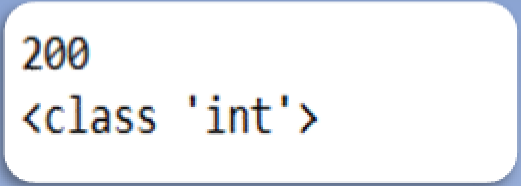
A variable is assigned using the = operator

Variable types cont..

- You can always check the type of a variable using the **type()** function.

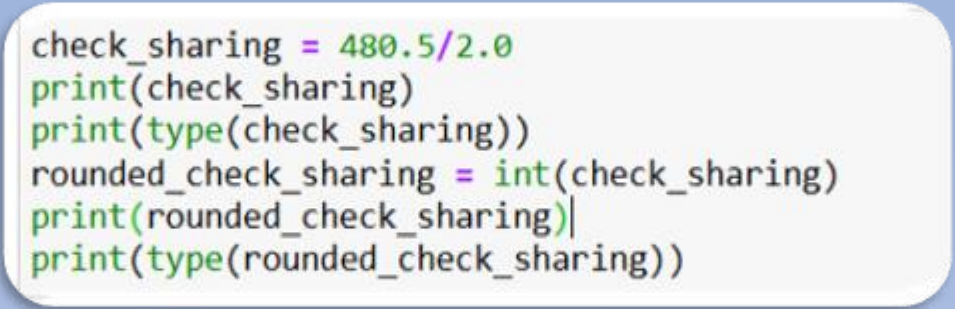


```
variable = 200
print(variable)
print(type(variable))
```

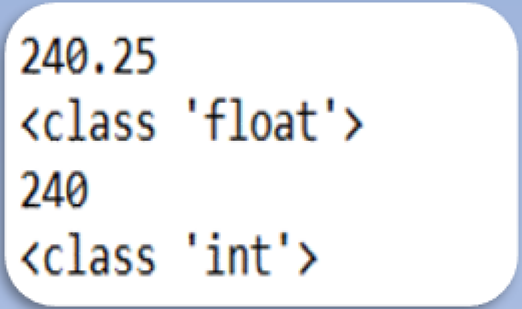


```
200
<class 'int'>
```

- Variables can be cast to a different type.



```
check_sharing = 480.5/2.0
print(check_sharing)
print(type(check_sharing))
rounded_check_sharing = int(check_sharing)
print(rounded_check_sharing)
print(type(rounded_check_sharing))
```



```
240.25
<class 'float'>
240
<class 'int'>
```

Numbers -> Integers & Floats

The arithmetic operators:

- Addition: +
- Subtract: -
- Multiplication: *
- Division: /
- Power: **
- Modulus: %

A quick note on the increment operator shorthand

Python has a common idiom that is not necessary, but which is used frequently and is therefore worth noting:

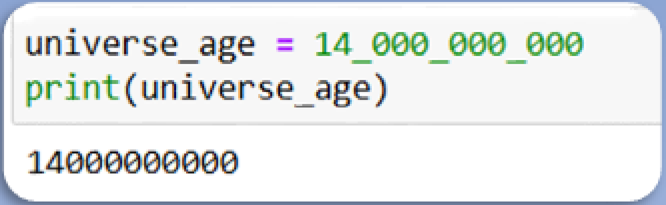
x += 1
Is the same as:
x = x + 1

This also works for other operators:

x += y adds y to the value of x
x *= y multiplies x by the value y
x -= y subtracts y from x
x /= y divides x by y

Underscores in Numbers & Multiple Assignment

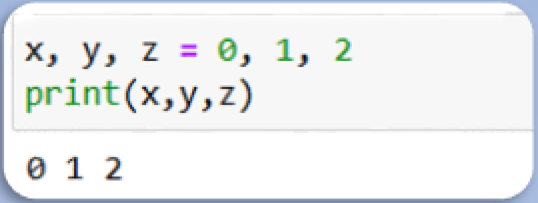
- When you’re writing long numbers, you can group digits using underscores to make large numbers more readable:



```
universe_age = 14_000_000_000
print(universe_age)
```

```
14000000000
```

- You can assign values to more than one variable using just a single line:

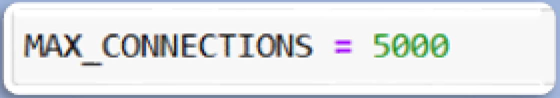


```
x, y, z = 0, 1, 2
print(x,y,z)
```

```
0 1 2
```

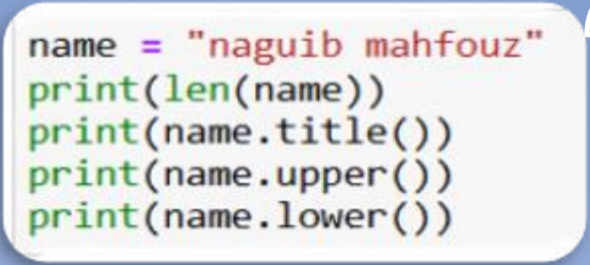
Constants

- When A constant is like a variable whose value stays the same throughout the life of a program.
- Python doesn’t have built-in constant types, but Python programmers use all capital letters to indicate a variable should be treated as a constant and never be changed:

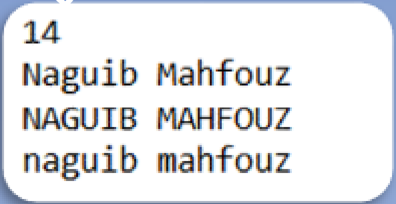


```
MAX_CONNECTIONS = 5000
```

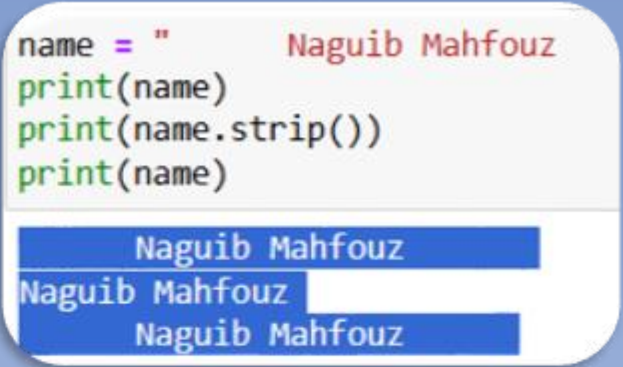
Operations on Strings



```python
name = "naguib mahfouz"
print(len(name))
print(name.title())
print(name.upper())
print(name.lower())
```

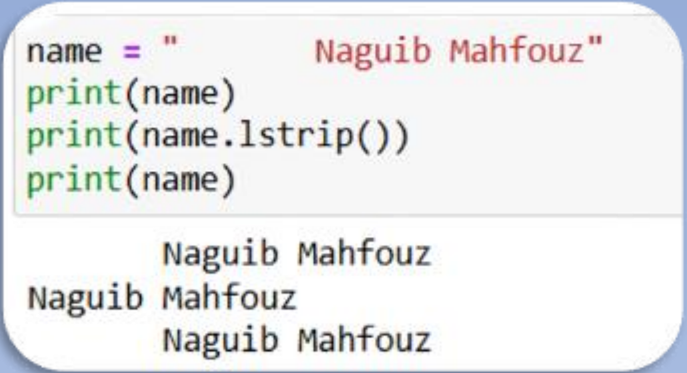


```
14
Naguib Mahfouz
NAGUIB MAHFOUZ
naguib mahfouz
```



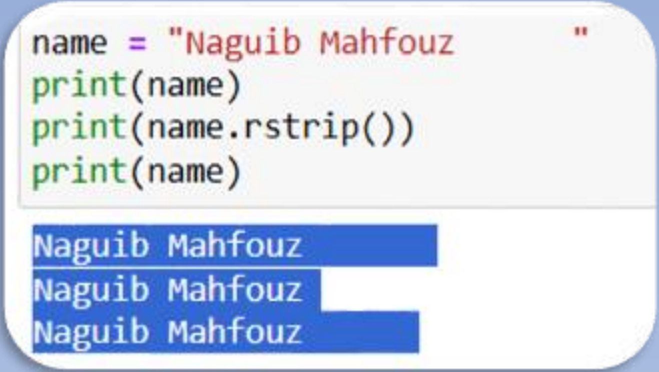
```python
name = "        Naguib Mahfouz
print(name)
print(name.strip())
print(name)
```

```
        Naguib Mahfouz
Naguib Mahfouz
        Naguib Mahfouz
```



```python
name = "        Naguib Mahfouz"
print(name)
print(name.lstrip())
print(name)
```

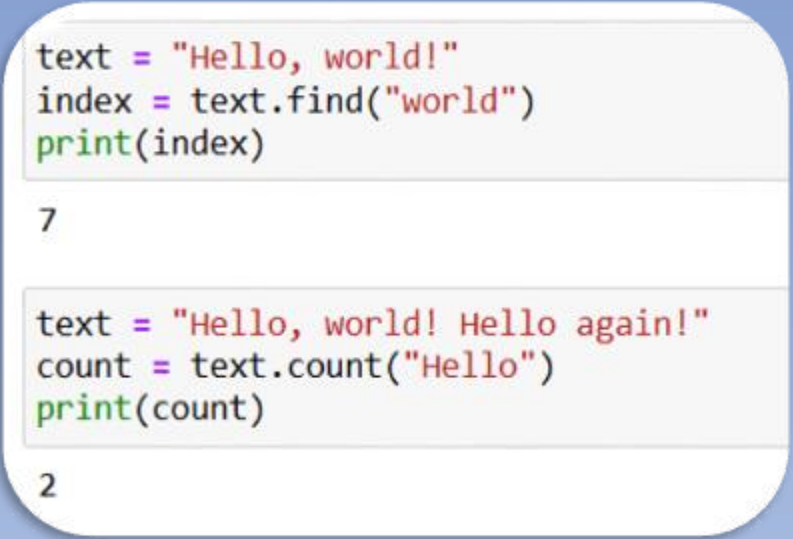
```
        Naguib Mahfouz
Naguib Mahfouz
        Naguib Mahfouz
```



```python
name = "Naguib Mahfouz        "
print(name)
print(name.rstrip())
print(name)
```

```
Naguib Mahfouz
Naguib Mahfouz
Naguib Mahfouz
```

Operations on Strings Cont..

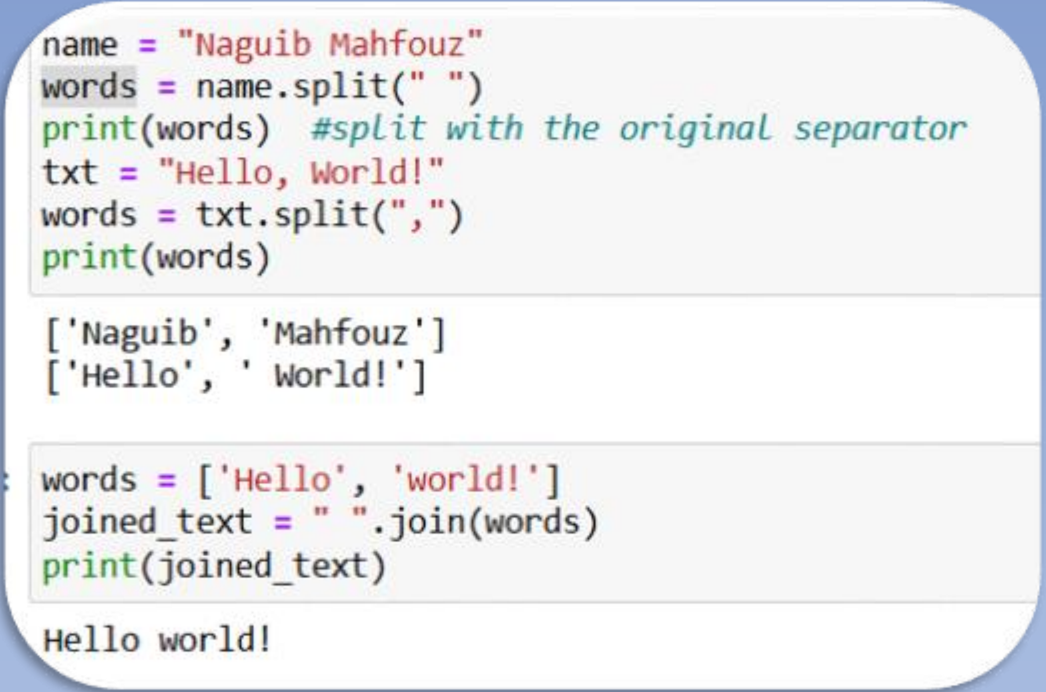


```
text = "Hello, world!"
index = text.find("world")
print(index)
```

```
7
```

```
text = "Hello, world! Hello again!"
count = text.count("Hello")
print(count)
```

```
2
```



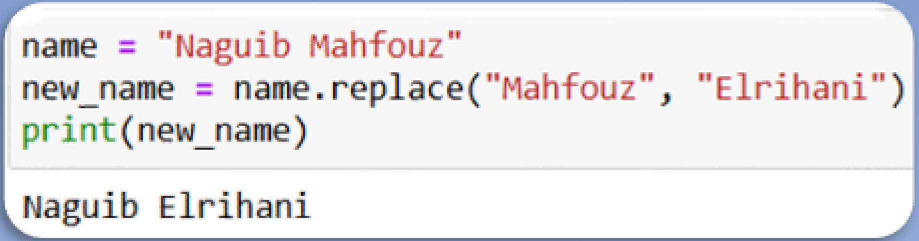
```
name = "Naguib Mahfouz"
words = name.split(" ")
print(words)  #split with the original separator
txt = "Hello, World!"
words = txt.split(",")
print(words)
```

```
['Naguib', 'Mahfouz']
['Hello', ' World!']
```

```
words = ['Hello', 'world!']
joined_text = " ".join(words)
print(joined_text)
```

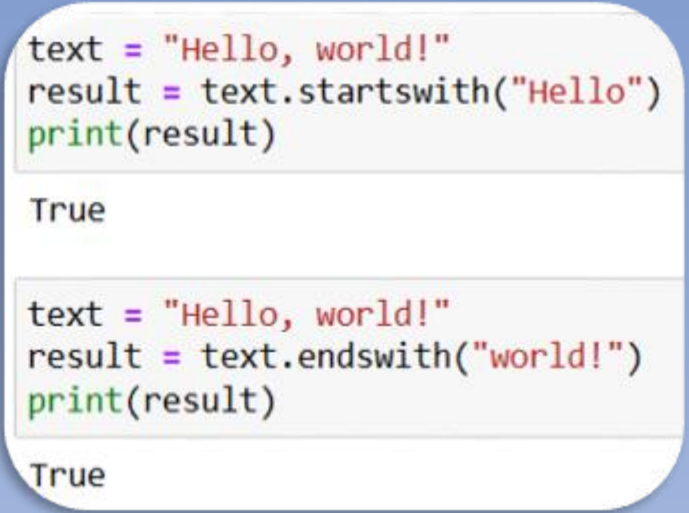
```
Hello world!
```

Operations on Strings Cont..



```
name = "Naguib Mahfouz"
new_name = name.replace("Mahfouz", "Elrihani")
print(new_name)
```

```
Naguib Elrihani
```

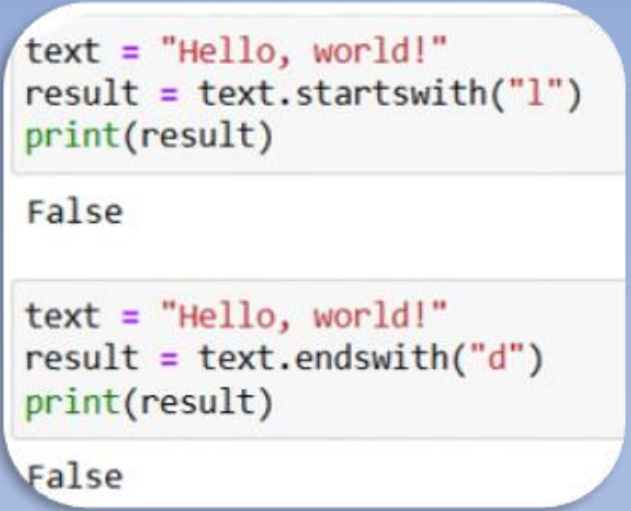


```
text = "Hello, world!"
result = text.startswith("Hello")
print(result)
```

```
True
```

```
text = "Hello, world!"
result = text.endswith("world!")
print(result)
```

```
True
```



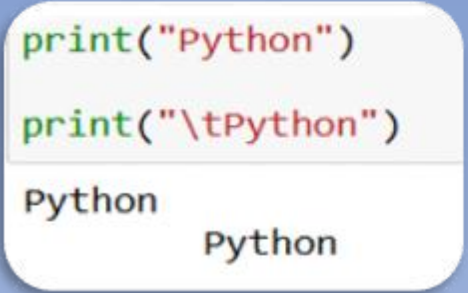
```
text = "Hello, world!"
result = text.startswith("l")
print(result)
```

```
False
```

```
text = "Hello, world!"
result = text.endswith("d")
print(result)
```

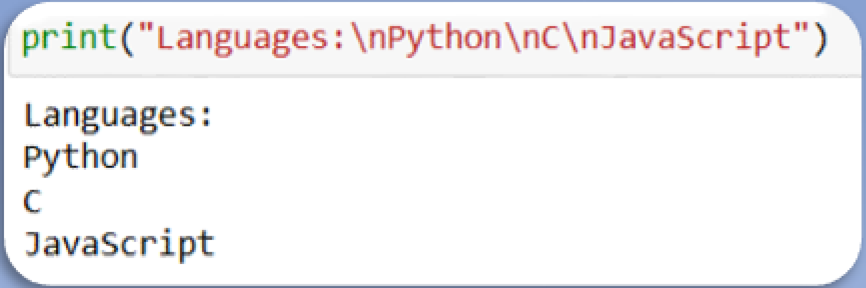
```
False
```

Adding Whitespace to Strings with Tabs or Newlines



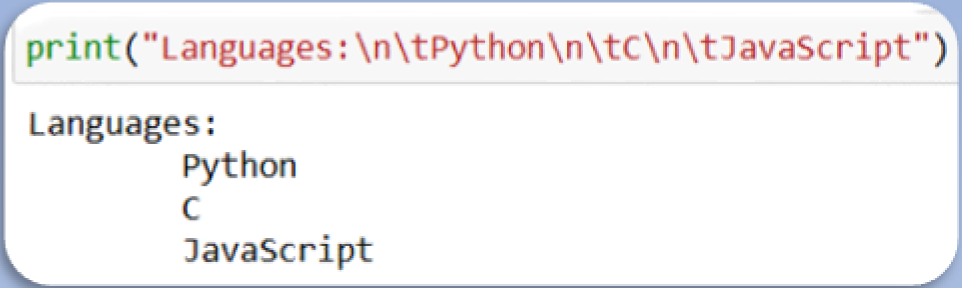
```
print("Python")

print("\tPython")
```

```
Python
	Python
```



```
print("Languages:\nPython\nC\nJavaScript")
```

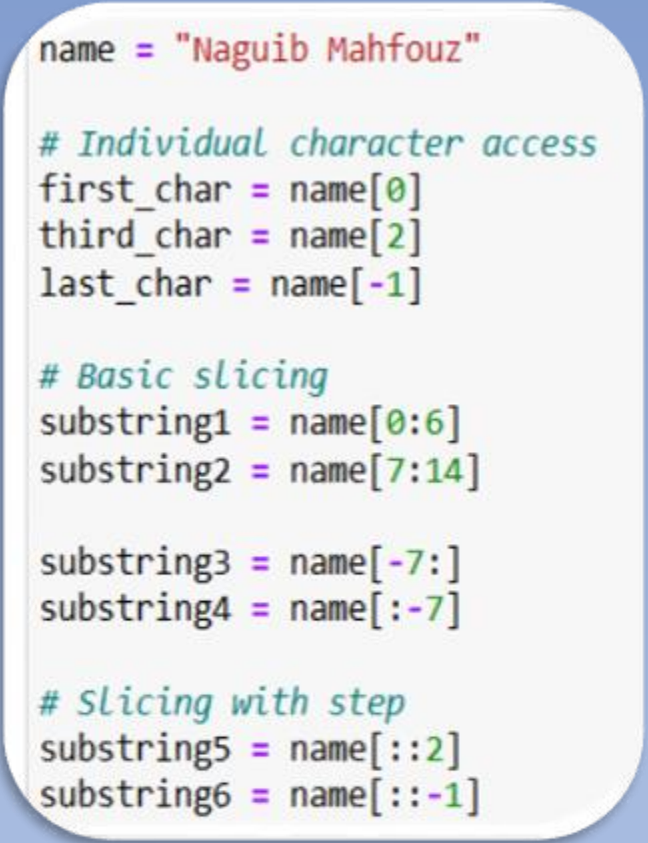
```
Languages:
Python
C
JavaScript
```



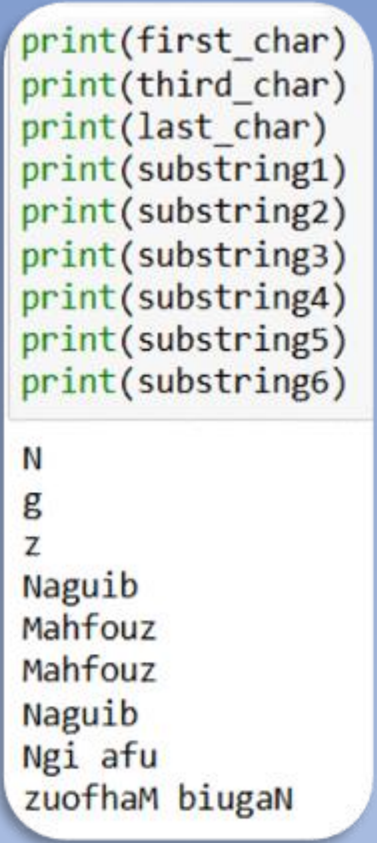
```
print("Languages:\n\tPython\n\tC\n\tJavaScript")
```

```
Languages:
	Python
	C
	JavaScript
```

Indexing & Slicing



```
name = "Naguib Mahfouz"

# Individual character access
first_char = name[0]
third_char = name[2]
last_char = name[-1]

# Basic slicing
substring1 = name[0:6]
substring2 = name[7:14]

substring3 = name[-7:]
substring4 = name[:-7]

# Slicing with step
substring5 = name[::2]
substring6 = name[::-1]
```



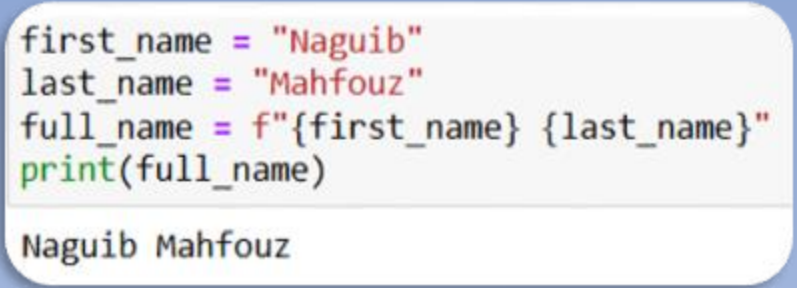
```
print(first_char)
print(third_char)
print(last_char)
print(substring1)
print(substring2)
print(substring3)
print(substring4)
print(substring5)
print(substring6)
```

```
N
g
z
Naguib
Mahfouz
Mahfouz
Naguib
Ngi afu
zuofhaM biugaN
```

Using Variables in Strings

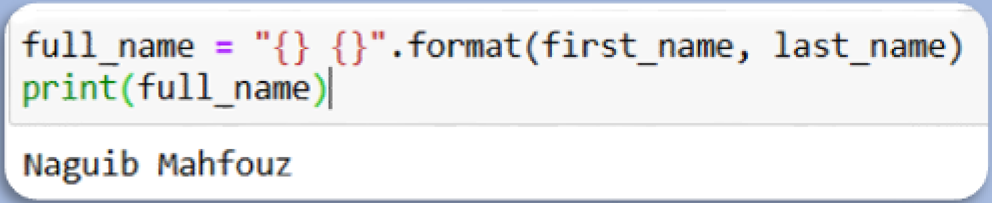
In some situations, you'll want to use a variable's value inside a string.

For example, you might want two variables to represent a first name and a last name respectively, and then want to combine those values to display someone's full name:



```python
first_name = "Naguib"
last_name = "Mahfouz"
full_name = f"{first_name} {last_name}"
print(full_name)
```

```
Naguib Mahfouz
```



```python
full_name = "{} {}".format(first_name, last_name)
print(full_name)
```

```
Naguib Mahfouz
```

04

Boolean & Comparison operators

Boolean & Comparison Operator

Boolean operators are useful when making conditional statements, we will cover these in-depth later.

and
or
not

Comparison Operators:

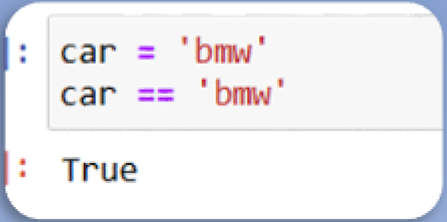
Greater than: >
Lesser than: <
Greater than or equal to: >=
Lesser than or equal to: <=
Is equal to: ==

05

If Statement

Conditional Tests

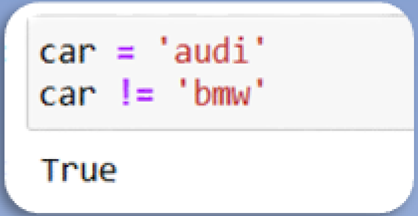
Checking for Equality



```
car = 'bmw'
car == 'bmw'
```

```
True
```

Checking for Inequality

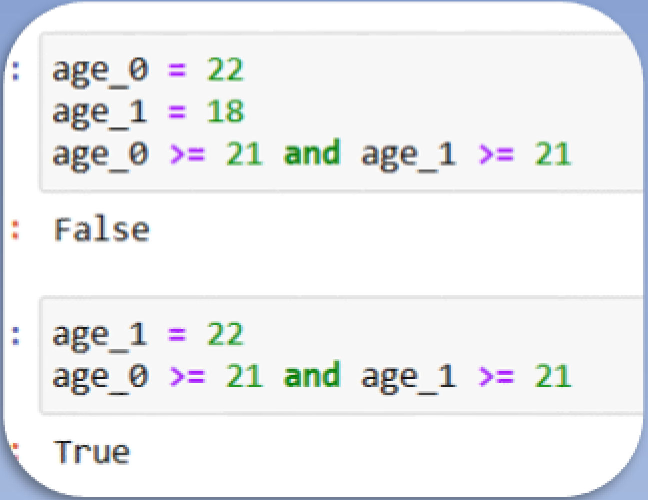


```
car = 'audi'
car != 'bmw'
```

```
True
```

Checking Multiple Conditions

Using and to Check Multiple Conditions



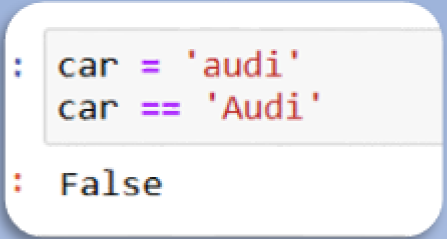
```
age_0 = 22
age_1 = 18
age_0 >= 21 and age_1 >= 21
```

```
False
```

```
age_1 = 22
age_0 >= 21 and age_1 >= 21
```

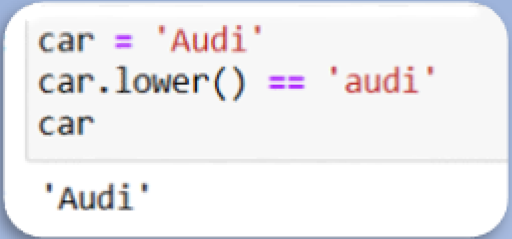
```
True
```

Ignoring Case When Checking for Equality



```
car = 'audi'
car == 'Audi'
```

```
False
```



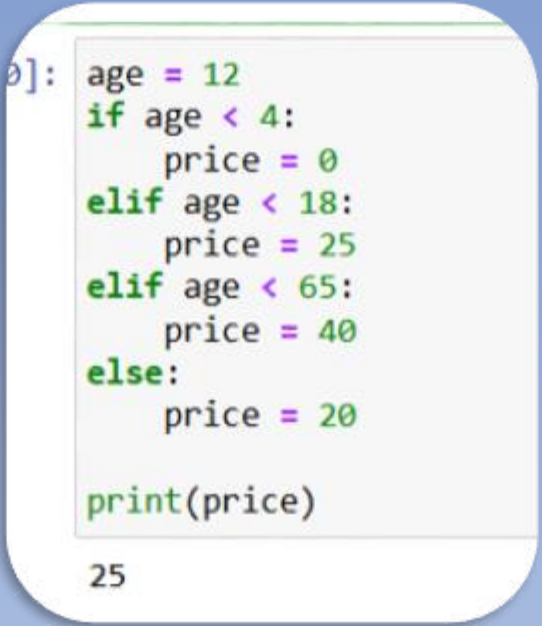
```
car = 'Audi'
car.lower() == 'audi'
car
```

```
'Audi'
```

if-else statement

For example, consider an amusement park that charges different rates for different age groups:

- Admission for anyone under age 4 is free.
- Admission for anyone between the ages of 4 and 18 is $25.
- Admission for anyone age 18 or older is $40.



```
]:  age = 12
    if age < 4:
        price = 0
    elif age < 18:
        price = 25
    elif age < 65:
        price = 40
    else:
        price = 20

    print(price)
```

```
25
```

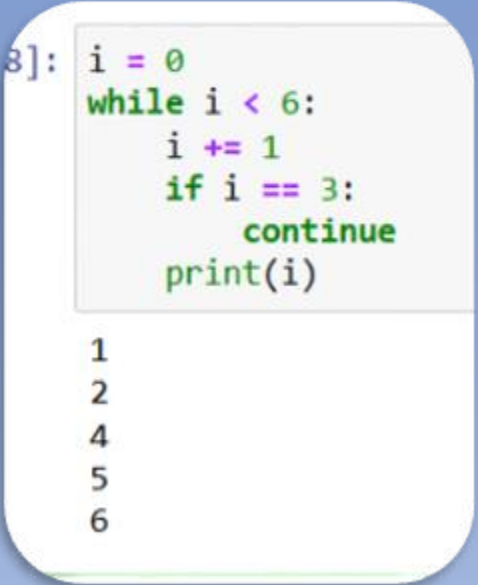
06

Loops

While loop

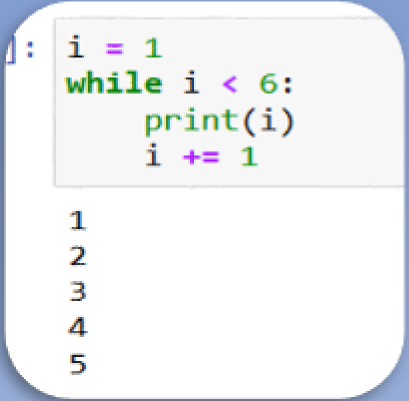
With the while loop we can execute a set of statements as long as a condition is true.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.



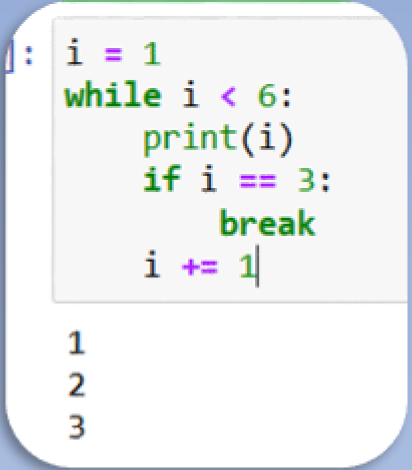
```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

```
1
2
4
5
6
```



```
i = 1
while i < 6:
    print(i)
    i += 1
```

```
1
2
3
4
5
```



```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

```
1
2
3
```

For loop

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.

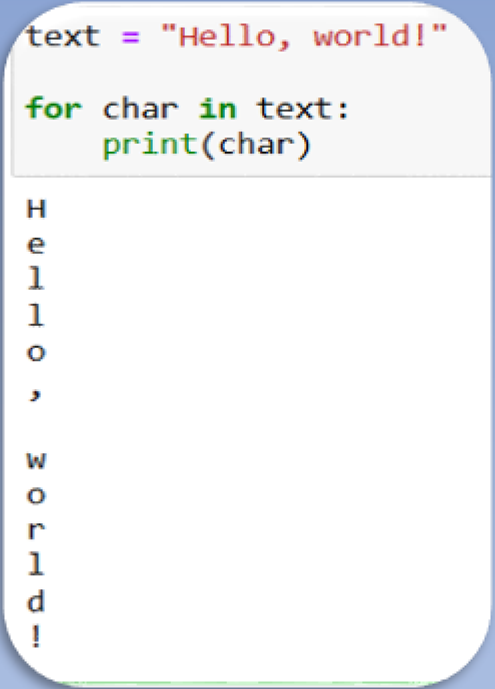
With the `for` loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
for variable in iterable:
    # Code block to execute
```

variable: This is the name you assign to the current item in the iteration. In each iteration, it takes on the value of the next item in the iterable.

Iterable: This can be any Python object that can return its elements one at a time, such as a list, tuple, string, or dictionary.

Code block: This is the indented code that will be executed for each item in the iterable.

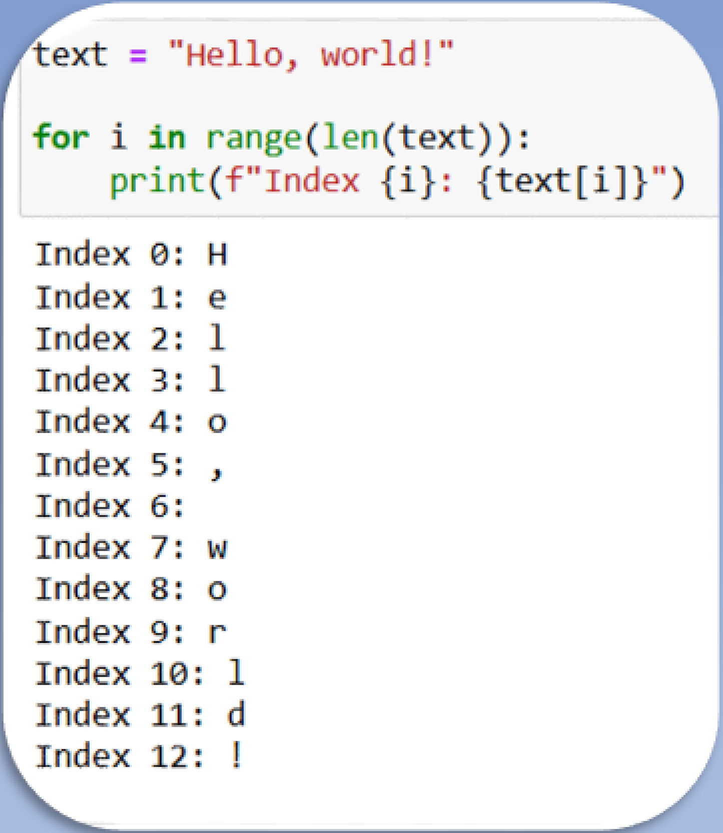


```
text = "Hello, world!"

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

```
H
e
l
l
o
,
 
w
o
r
l
d
!
```

Range

The range(start, stop, step) function is often used to define x. The starting value is defined by start, the final value is defined by stop – 1, and the magnitude at which x changes between loops is defined by step



```python
text = "Hello, world!"

for i in range(len(text)):
    print(f"Index {i}: {text[i]}")
```

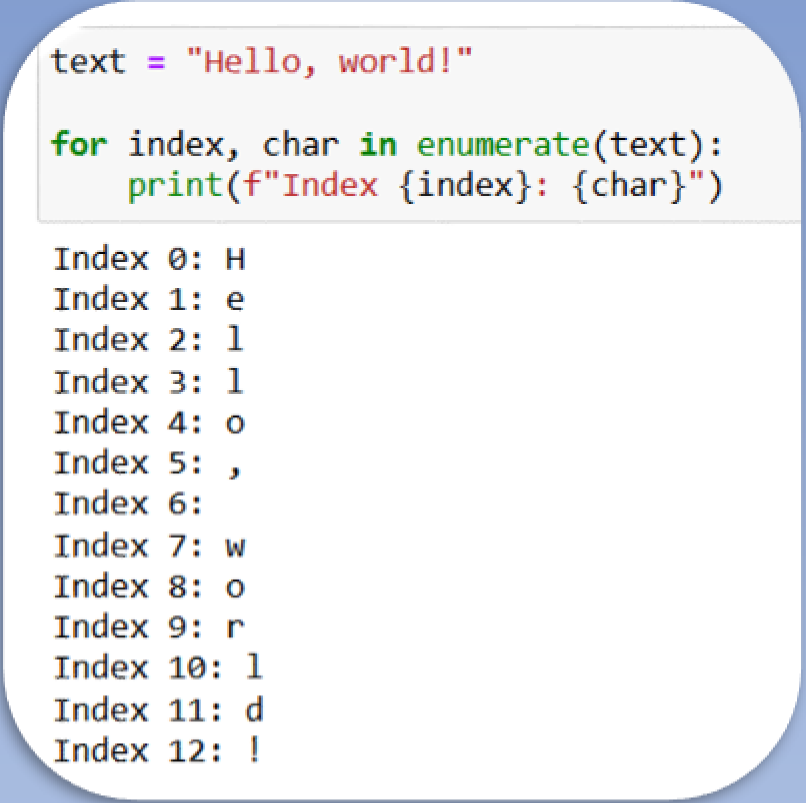
```
Index 0: H
Index 1: e
Index 2: l
Index 3: l
Index 4: o
Index 5: ,
Index 6:  
Index 7: w
Index 8: o
Index 9: r
Index 10: l
Index 11: d
Index 12: !
```

Enumerate

The enumerate() function in Python is used to iterate over a sequence (like a list or a string) while keeping track of the index of the current item. This is especially useful when you need both the item and its index during iteration.

```
for index, value in enumerate(iterable):
    # Code block to execute
```

index: This will hold the current index of the item in the iterable.
value: This will hold the current item from the iterable.
iterable: This can be any sequence, such as a list, tuple, or string.



```
text = "Hello, world!"

for index, char in enumerate(text):
    print(f"Index {index}: {char}")
```

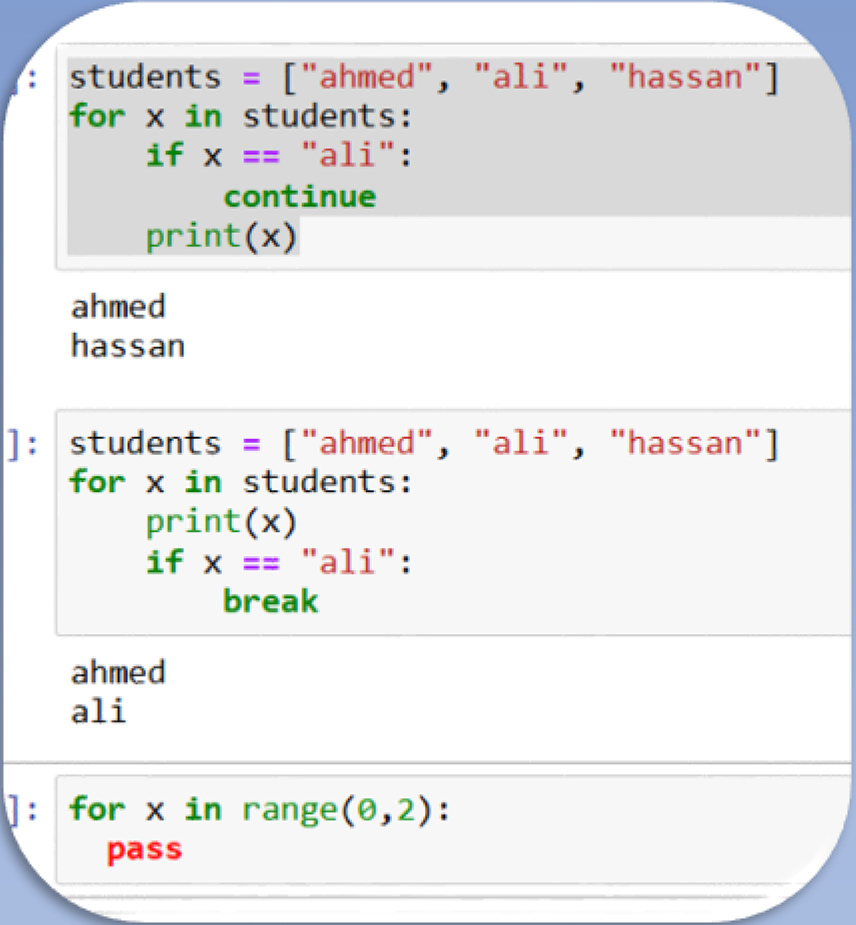
```
Index 0: H
Index 1: e
Index 2: l
Index 3: l
Index 4: o
Index 5: ,
Index 6:  
Index 7: w
Index 8: o
Index 9: r
Index 10: l
Index 11: d
Index 12: !
```

Continue Break and Pass Statement

The continue Statement:
With the continue statement we can stop the current iteration of the loop, and continue with the next

The break Statement
With the break statement we can stop the loop before it has looped through all the items

The pass Statement
for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.



```
students = ["ahmed", "ali", "hassan"]
for x in students:
    if x == "ali":
        continue
    print(x)
```

```
ahmed
hassan
```

```
students = ["ahmed", "ali", "hassan"]
for x in students:
    print(x)
    if x == "ali":
        break
```

```
ahmed
ali
```

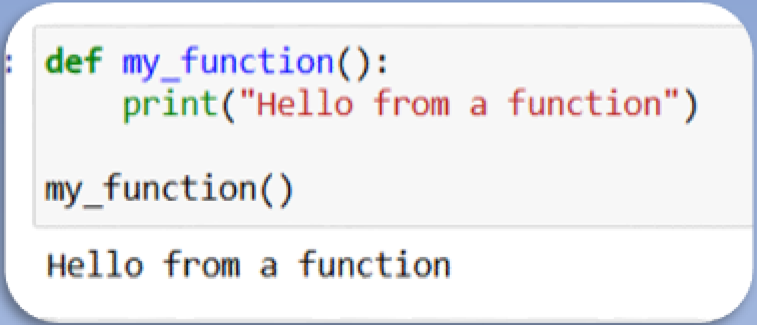
```
for x in range(0,2):
  pass
```

07

Functions

Creating a Function

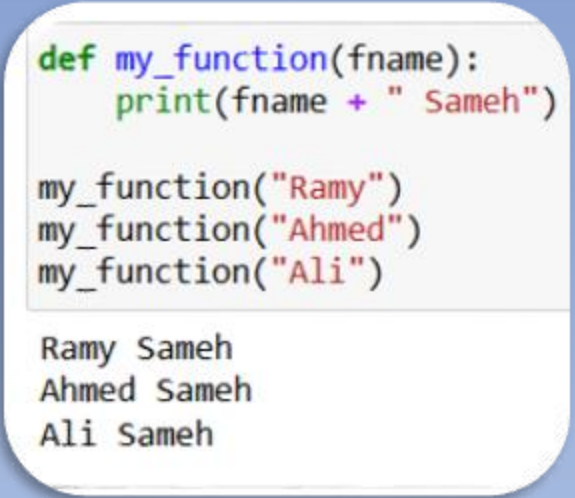
In Python a function is defined using the def keyword:



```
def my_function():
    print("Hello from a function")

my_function()
```

```
Hello from a function
```

Adding Arguments:



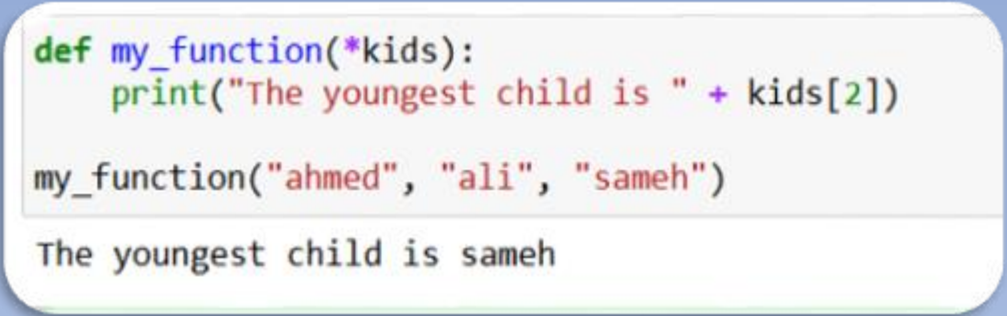
```
def my_function(fname):
    print(fname + " Sameh")

my_function("Ramy")
my_function("Ahmed")
my_function("Ali")
```

```
Ramy Sameh
Ahmed Sameh
Ali Sameh
```

Arbitrary Arguments, *args & Keyword Arguments

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

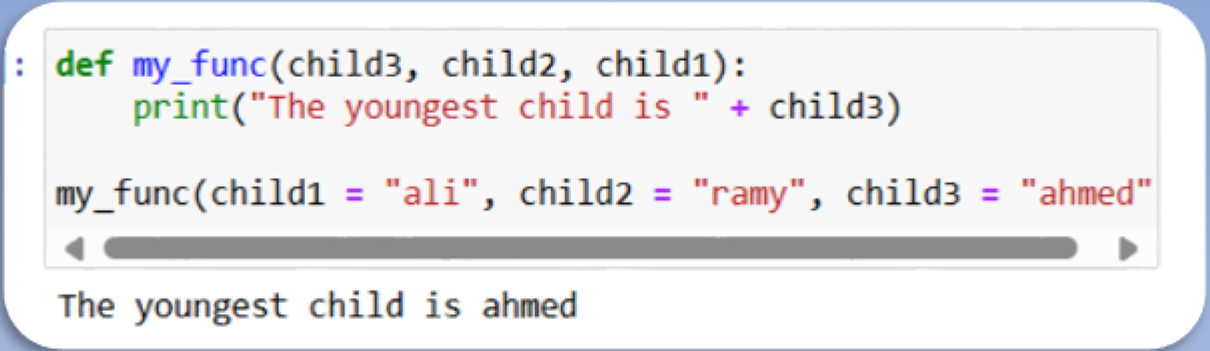


```
def my_function(*kids):
    print("The youngest child is " + kids[2])

my_function("ahmed", "ali", "sameh")
```

```
The youngest child is sameh
```

Keyword Arguments

You can send arguments with the *key* = *value* syntax.
This way the order of the arguments does not matter.



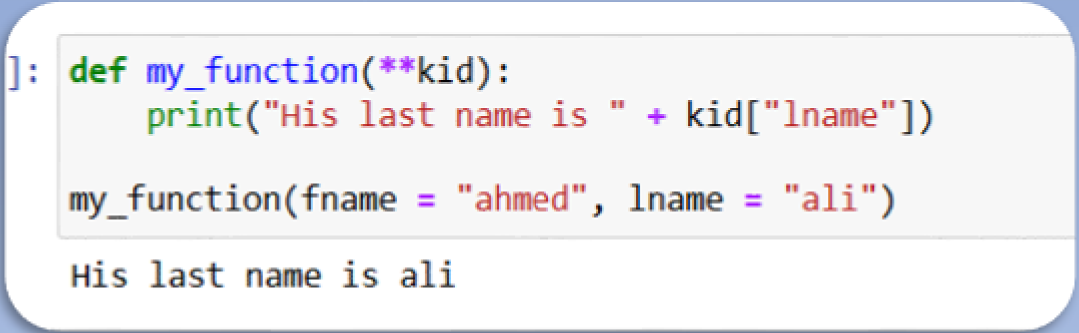
```
def my_func(child3, child2, child1):
    print("The youngest child is " + child3)

my_func(child1 = "ali", child2 = "ramy", child3 = "ahmed"
```

```
The youngest child is ahmed
```

Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

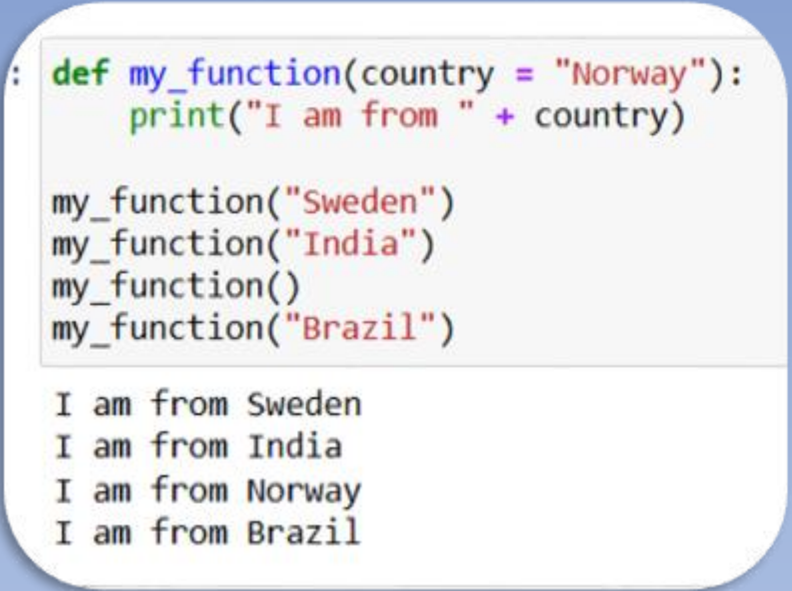


```
]: def my_function(**kid):
       print("His last name is " + kid["lname"])

   my_function(fname = "ahmed", lname = "ali")
```

```
His last name is ali
```

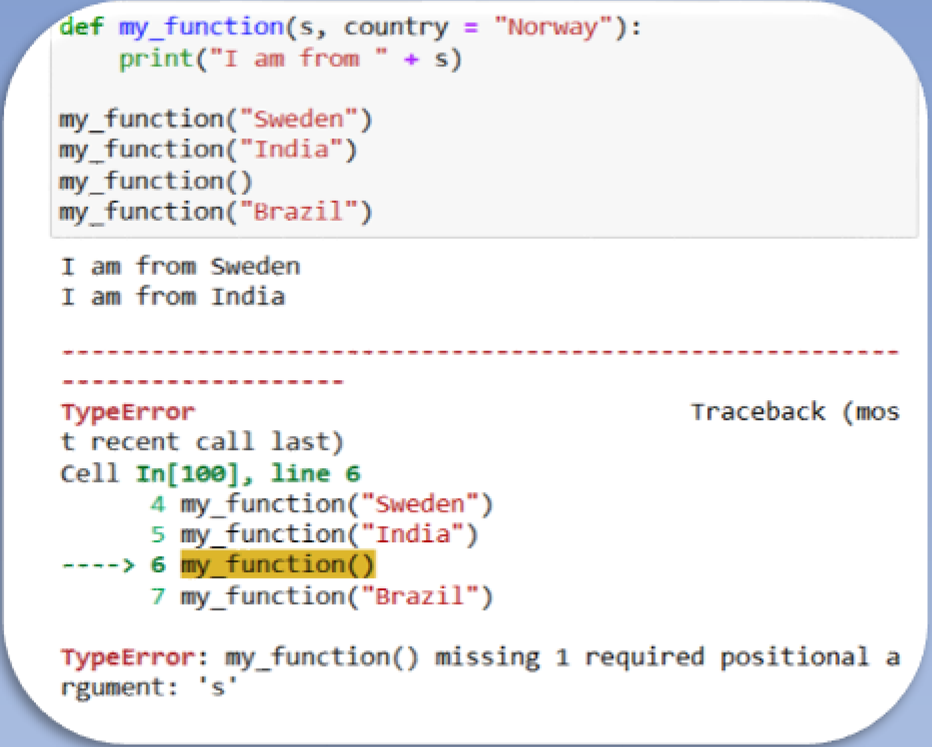
Default Parameter Value

If we call the function without argument, it uses the default value:



```
def my_function(country = "Norway"):
    print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

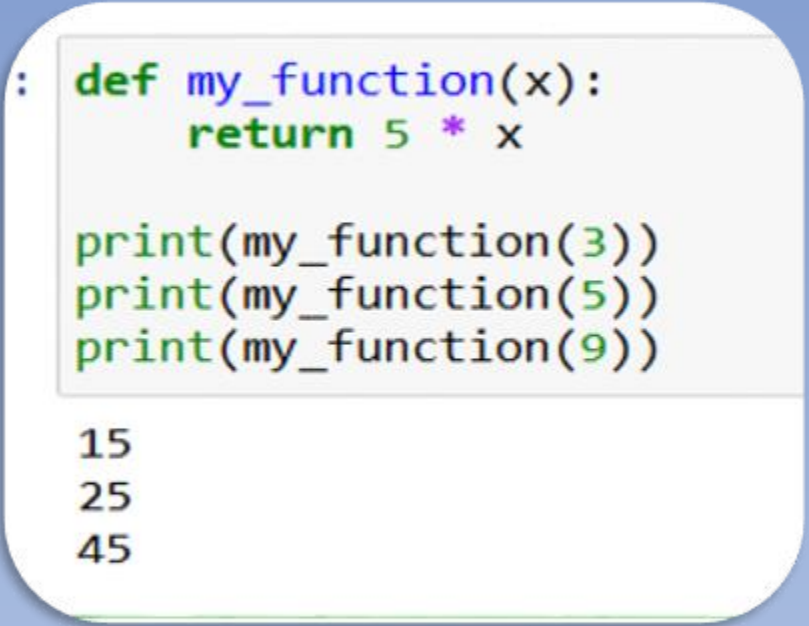
```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```



```
def my_function(s, country = "Norway"):
    print("I am from " + s)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India

---------------------------------------------------------------------------
TypeError                                 Traceback (mos
t recent call last)
Cell In[100], line 6
      4 my_function("Sweden")
      5 my_function("India")
----> 6 my_function()
      7 my_function("Brazil")

TypeError: my_function() missing 1 required positional a
rgument: 's'
```

Return value



```
def my_function(x):
    return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

Lambda Expression

A lambda function is a small anonymous function.

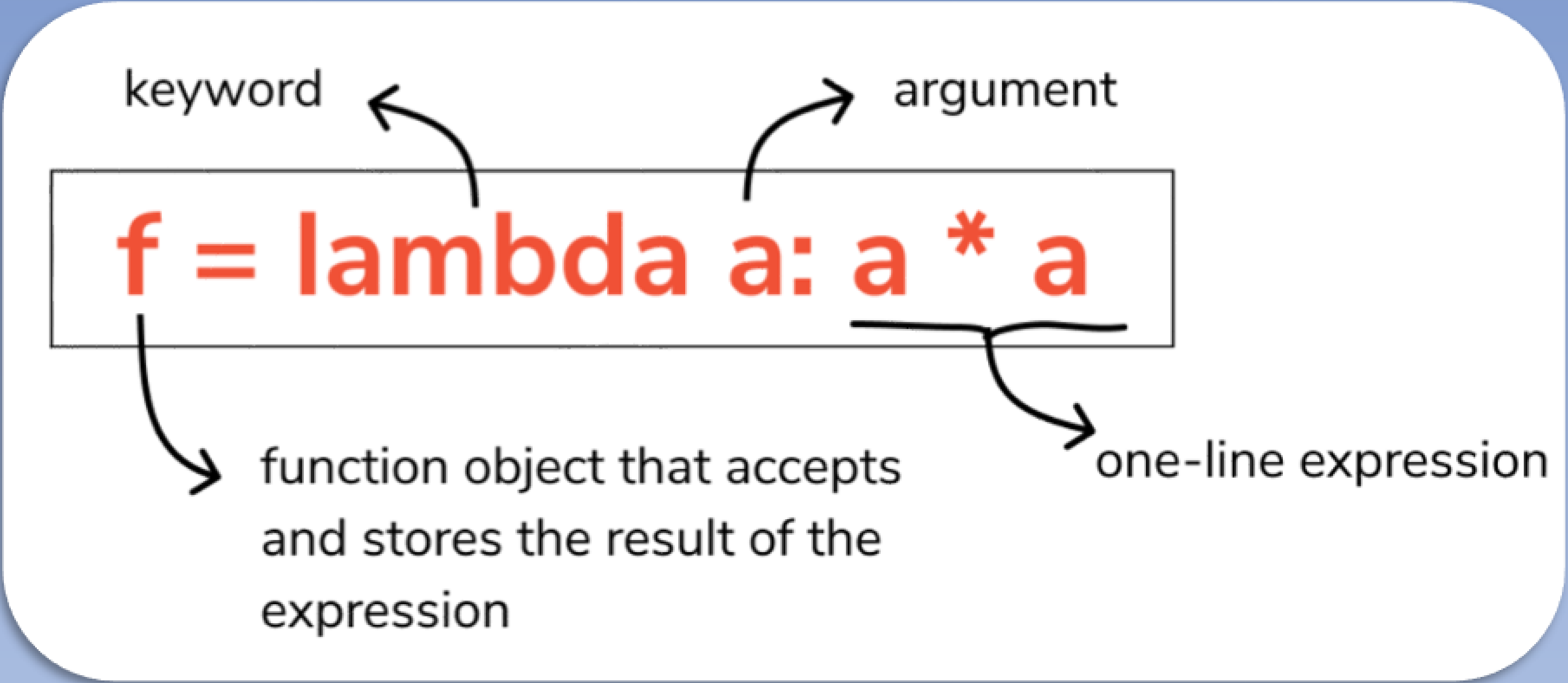
A lambda function can take any number of arguments, but can only have one expression.

Syntax

```
lambda arguments : expression
```

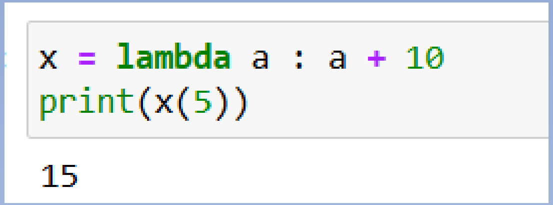
The expression is executed and the result is returned:

Lambda Expression


keyword
argument
f = lambda a: a * a
function object that accepts and stores the result of the expression
one-line expression

Lambda Expression Cont..

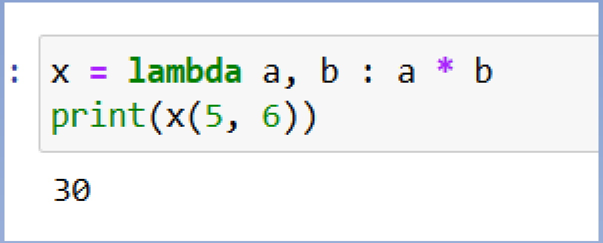
Example
Add 10 to argument a, and return the result:



```
x = lambda a : a + 10
print(x(5))
```

```
15
```

Multiply argument a with argument b and return the result:



```
x = lambda a, b : a * b
print(x(5, 6))
```

```
30
```

Thank You