Catching up On Python

Ines Ben Amor

What you will learn:

- What is an IDE
- What is a console and its main controls?
- Making conditions
- Make a FOR loop
- Make a WHILE

WHAT YOU W

- What is
- What is
- Making (
- Make a |
- Make a \

Code Logistics 🔨

CONSOLE OR TERMINAL

`pwd` (`cd` sur Windows)

`ls` **(or `dir` on Windows)**

`cd ..`
`open nomdufichier` ou (`nomdufichier` on Windows)

`clear` (`cls` **on Windows)**
touch nomdufichier.py (echo > nomdufichier.py on Windows)

mkdir nomdudossier

The REPL (Read Evaluate Print Loop)

$`python`

Et pour en sortir, écrire `quit()`

Code Editor or Text Editor

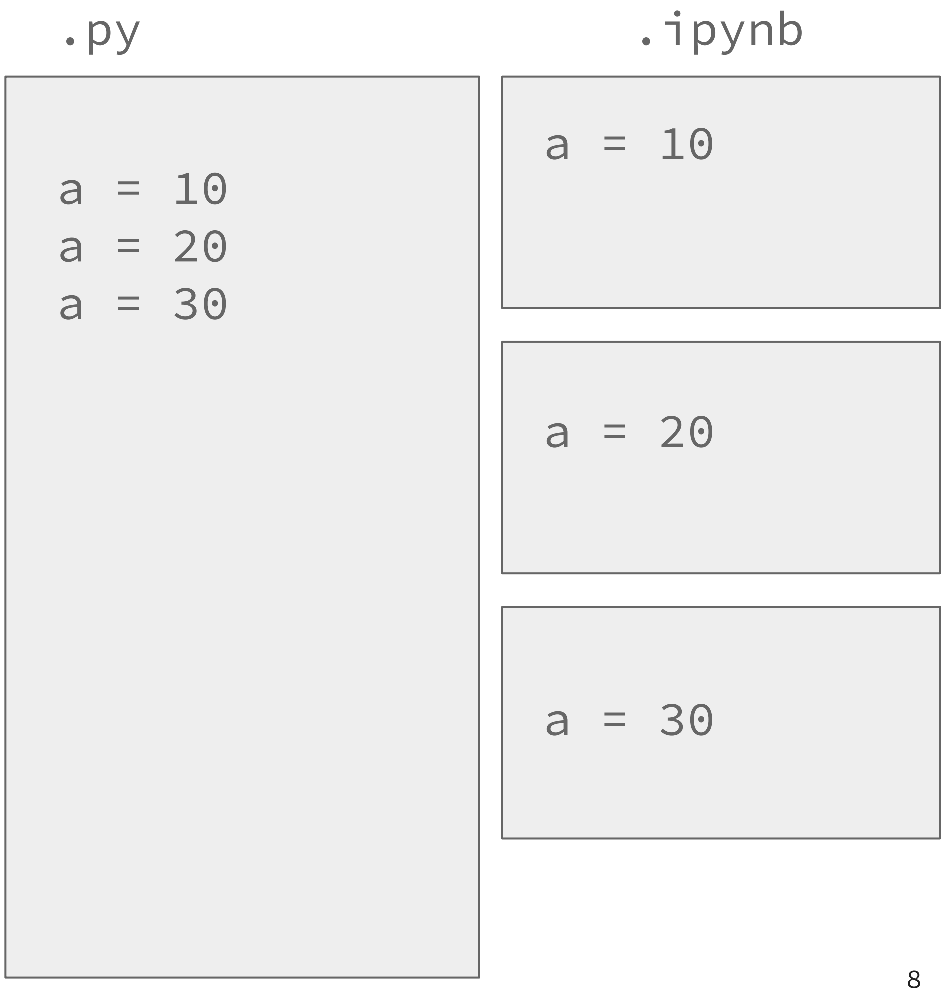
A text editor is where you'll put all your script.

Famous example: Notepad++

Files and Extensions

`.py` : **Python Script / file**

`.ipynb` : **Python Notebook**



```
a = 10
a = 20
a = 30
```

```
a = 10
```

```
a = 20
```

```
a = 30
```

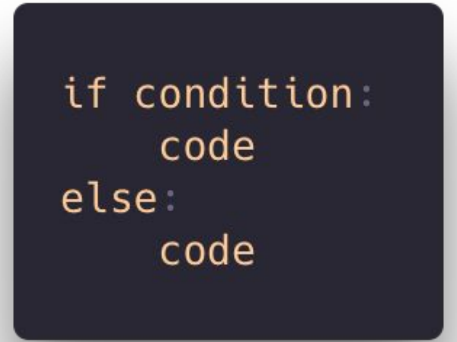
IDE = Integrated Development Environment

IDE = Console + Text Editor + Bonus

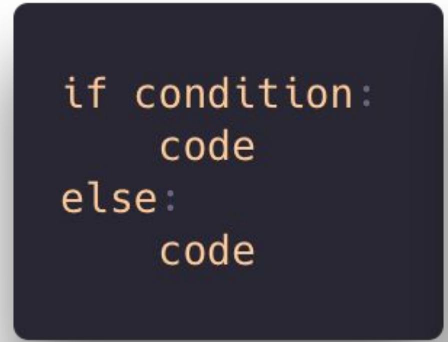
- **VS Code** (for Developers & Data people)
- **Jupyter vs Google Colab** (for Data people)
- **JULIE** (workspace in which Jupyter and all the tools needed to do Data tasks are installed)

Coding 👩‍💻

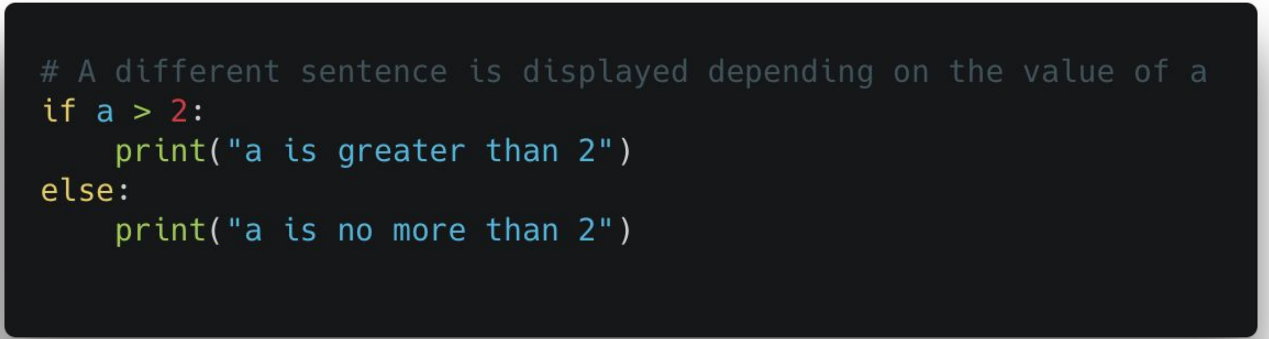
Building A Condition: if else


```
if condition:
    code
else:
    code
```

Building A Condition: If Else



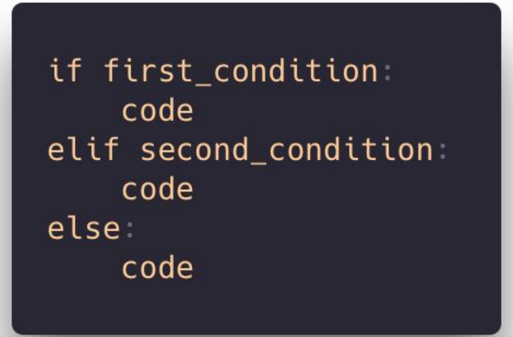
```
if condition:
     code
else:
     code
```



```
# A different sentence is displayed depending on the value of a
if a > 2:
    print("a is greater than 2")
else:
    print("a is no more than 2")
```

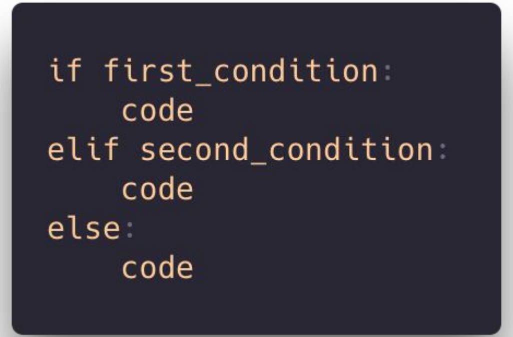
Ines Ben Amor

Building A Condition: If Elif Else

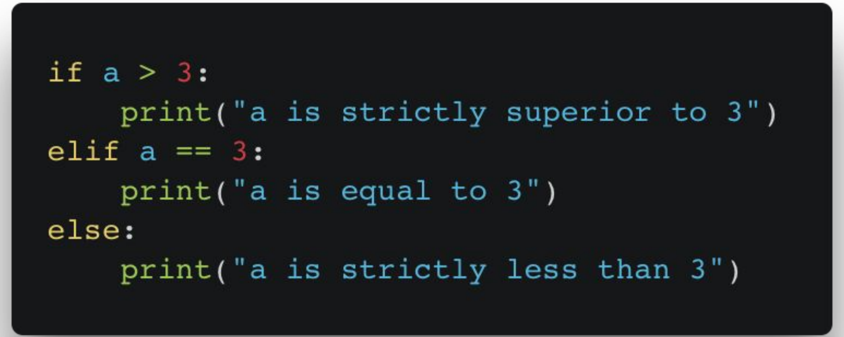


```
if first_condition:
    code
elif second_condition:
    code
else:
    code
```

Building A Condition: If Elif Else



```
if first_condition:
    code
elif second_condition:
    code
else:
    code
```



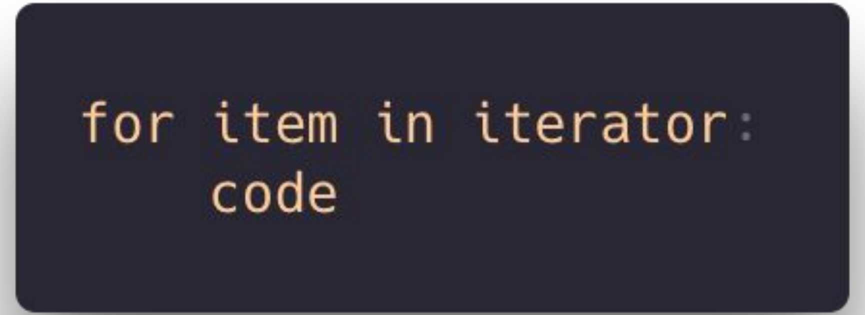
```
if a > 3:
    print("a is strictly superior to 3")
elif a == 3:
    print("a is equal to 3")
else:
    print("a is strictly less than 3")
```

Ines Ben Amor

Operators

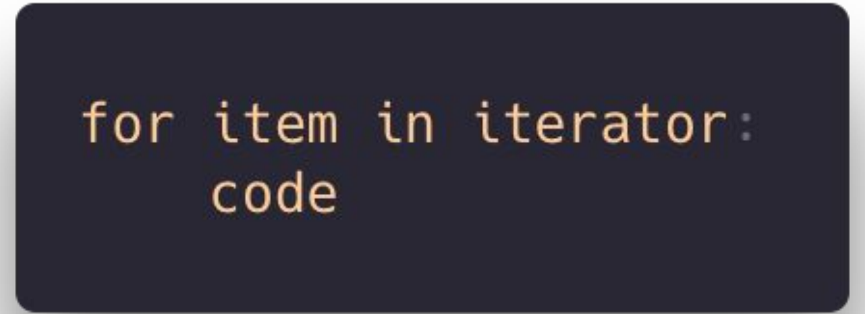
Operator	Meaning
`>`	Strictly superior
`<`	Strictly inferior
`>=`	Superior or equal
`<=`	Inferior or equal
`==`	Equal to (be careful to set the double equal otherwise it's as if you were assigning a new value to a variable)
`!=` (or `<>`)	Different from

Building A Loop: For

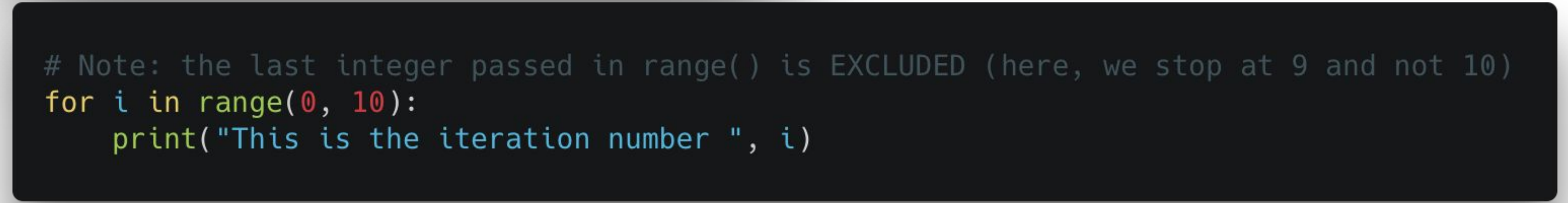


```
for item in iterator:
    code
```

Building a Loop: For

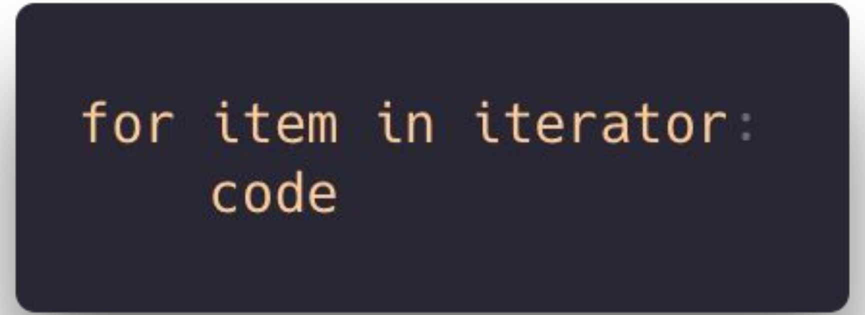


```
for item in iterator:
    code
```

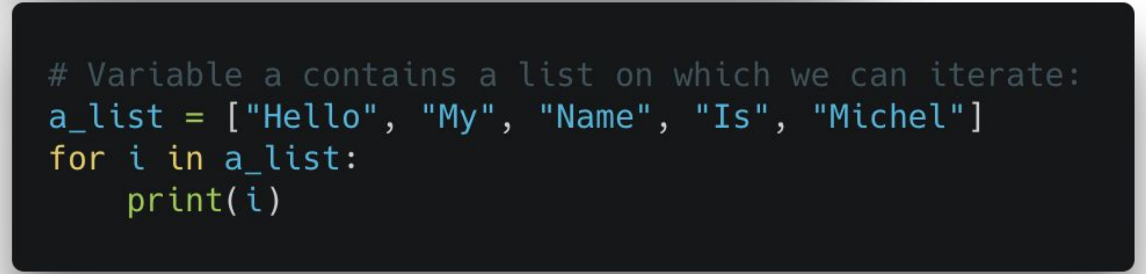


```
# Note: the last integer passed in range() is EXCLUDED (here, we stop at 9 and not 10)
for i in range(0, 10):
    print("This is the iteration number ", i)
```

Building a Loop: For

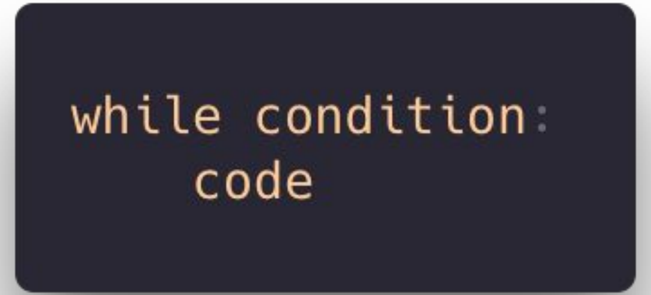


```
for item in iterator:
    code
```



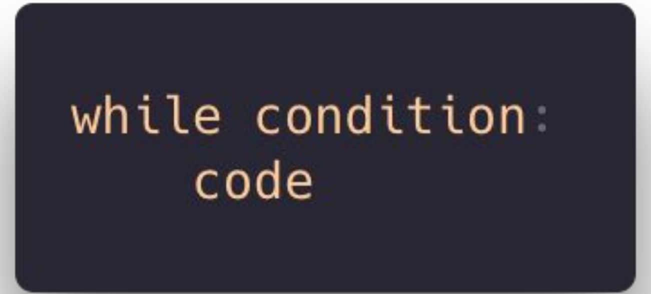
```python
# Variable a contains a list on which we can iterate:
a_list = ["Hello", "My", "Name", "Is", "Michel"]
for i in a_list:
    print(i)
```

Building A Loop: While

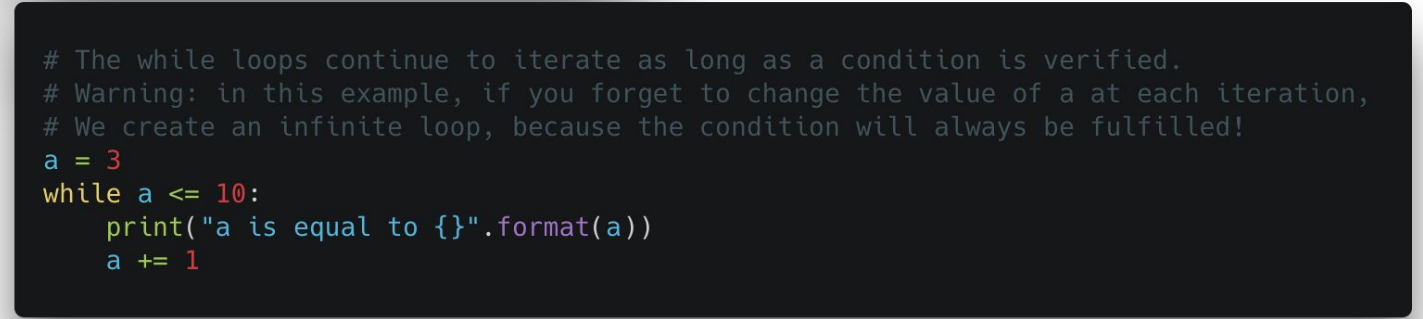


```
while condition:
    code
```

Building a Loop: While



```
while condition:
    code
```



```
# The while loops continue to iterate as long as a condition is verified.
# Warning: in this example, if you forget to change the value of a at each iteration,
# We create an infinite loop, because the condition will always be fulfilled!
a = 3
while a <= 10:
    print("a is equal to {}".format(a))
    a += 1
```

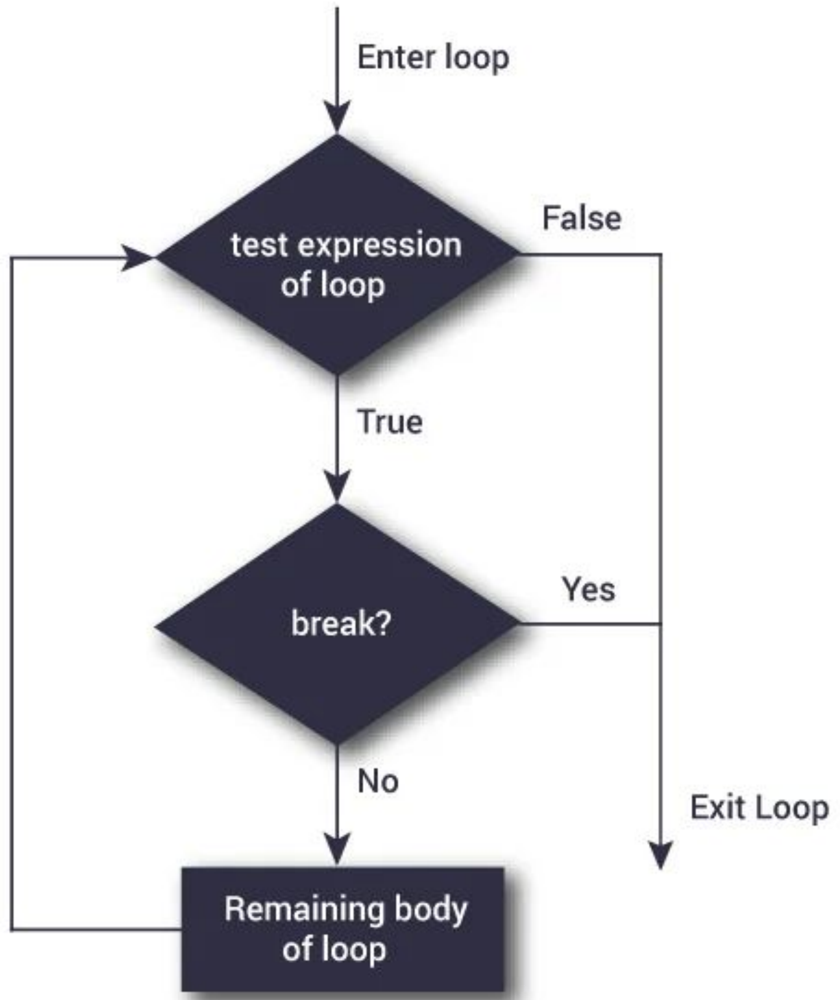
FOR != WHILE

FOR != WHILE

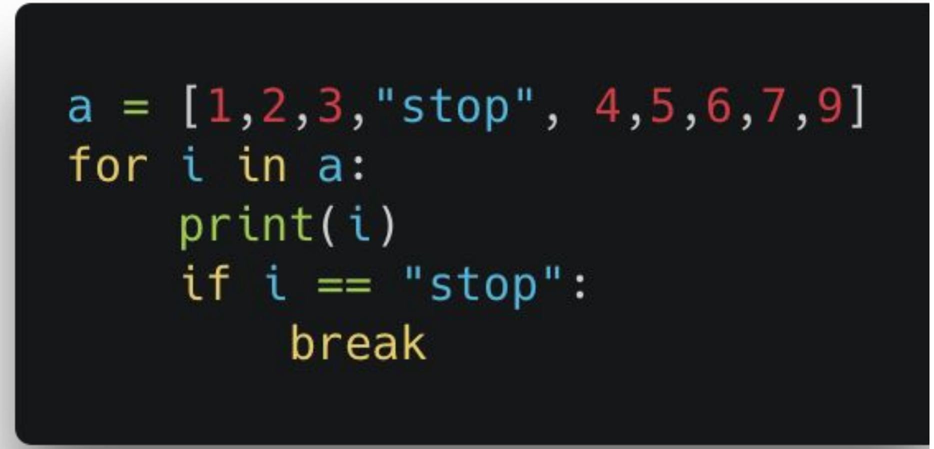
FOR : You know how many times to iterate

WHILE : You don’t know how many times to iterate

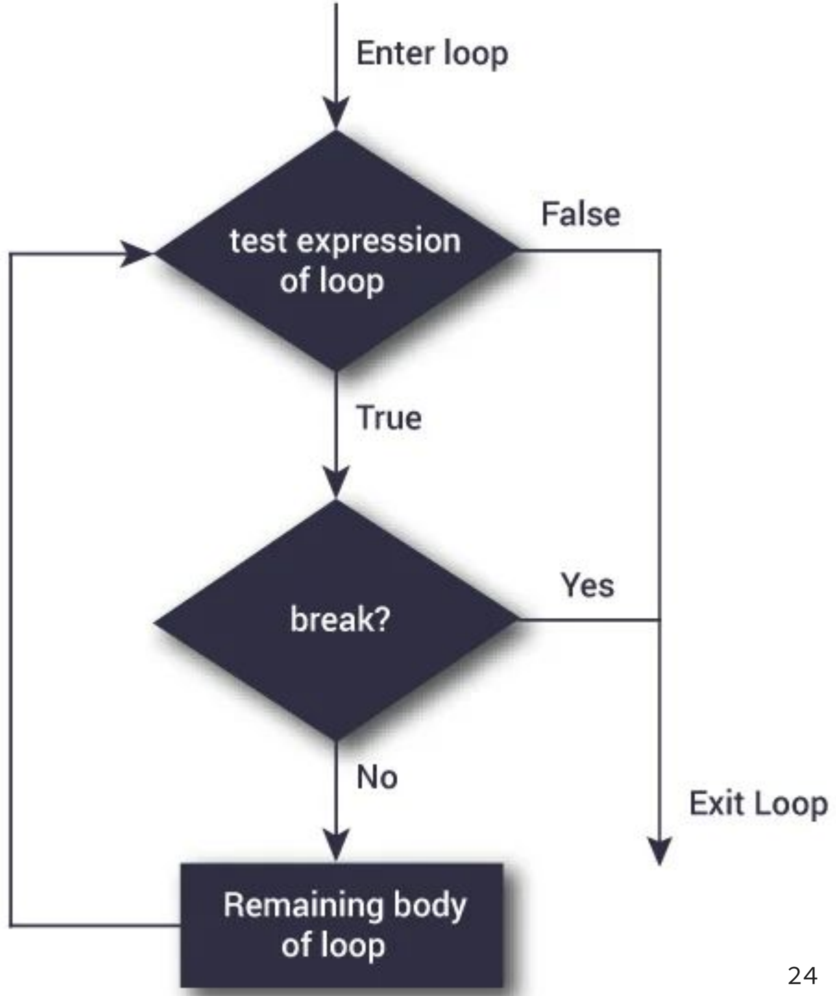
Leave a Loop


Enter loop
test expression of loop
False
True
break?
Yes
No
Exit Loop
Remaining body of loop

Leave a Loop

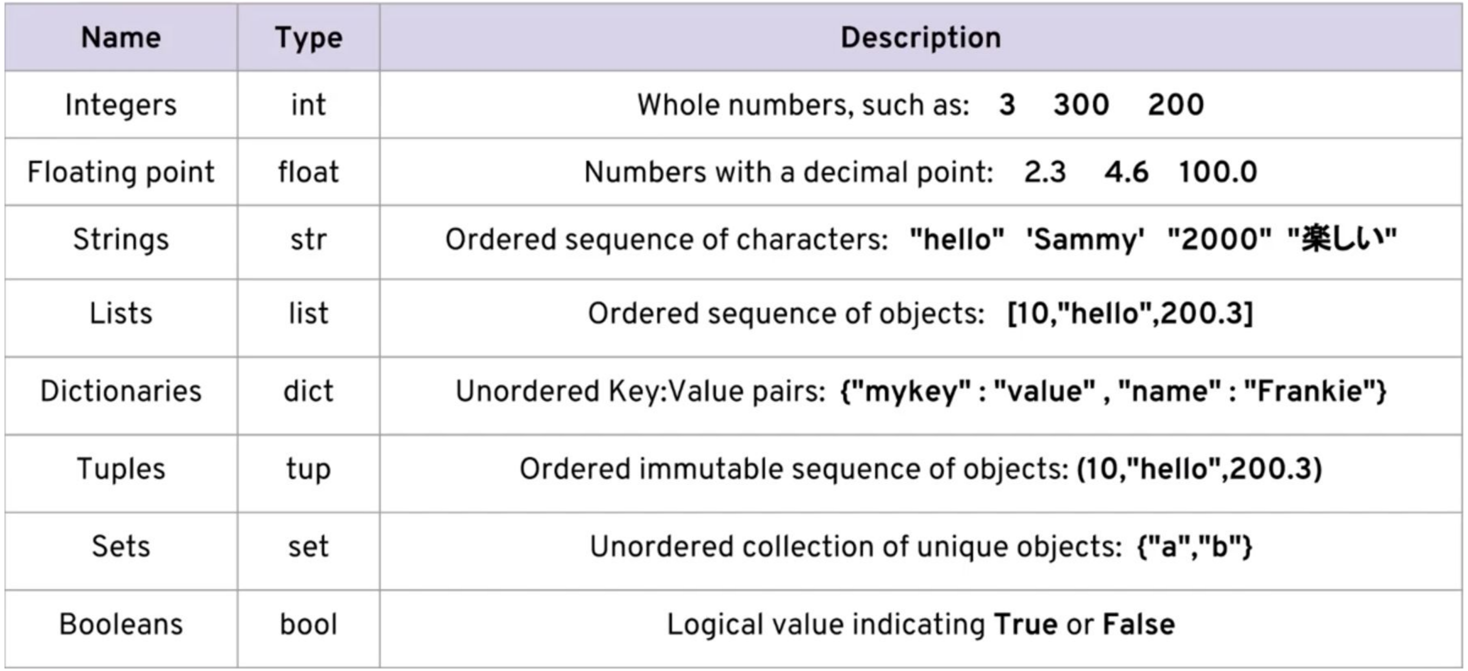


```python
a = [1,2,3,"stop", 4,5,6,7,9]
for i in a:
    print(i)
    if i == "stop":
        break
```



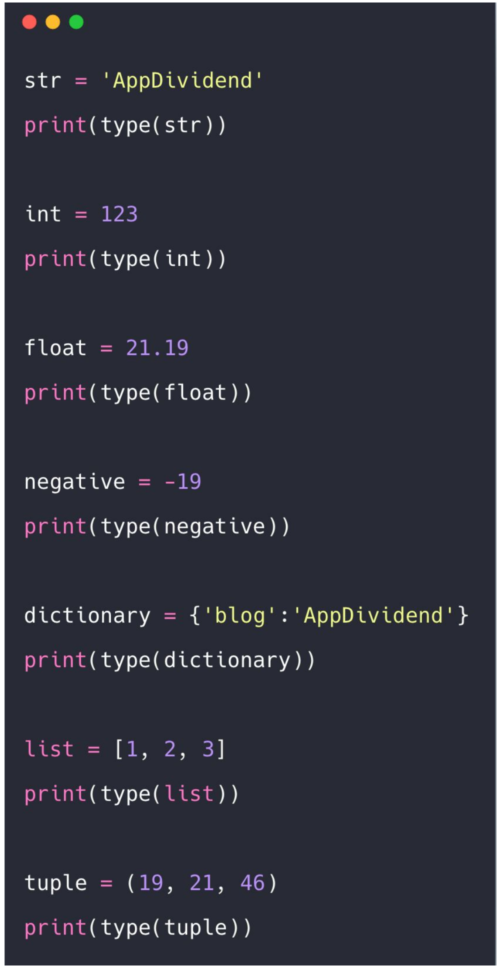
Data Types

Data types summary



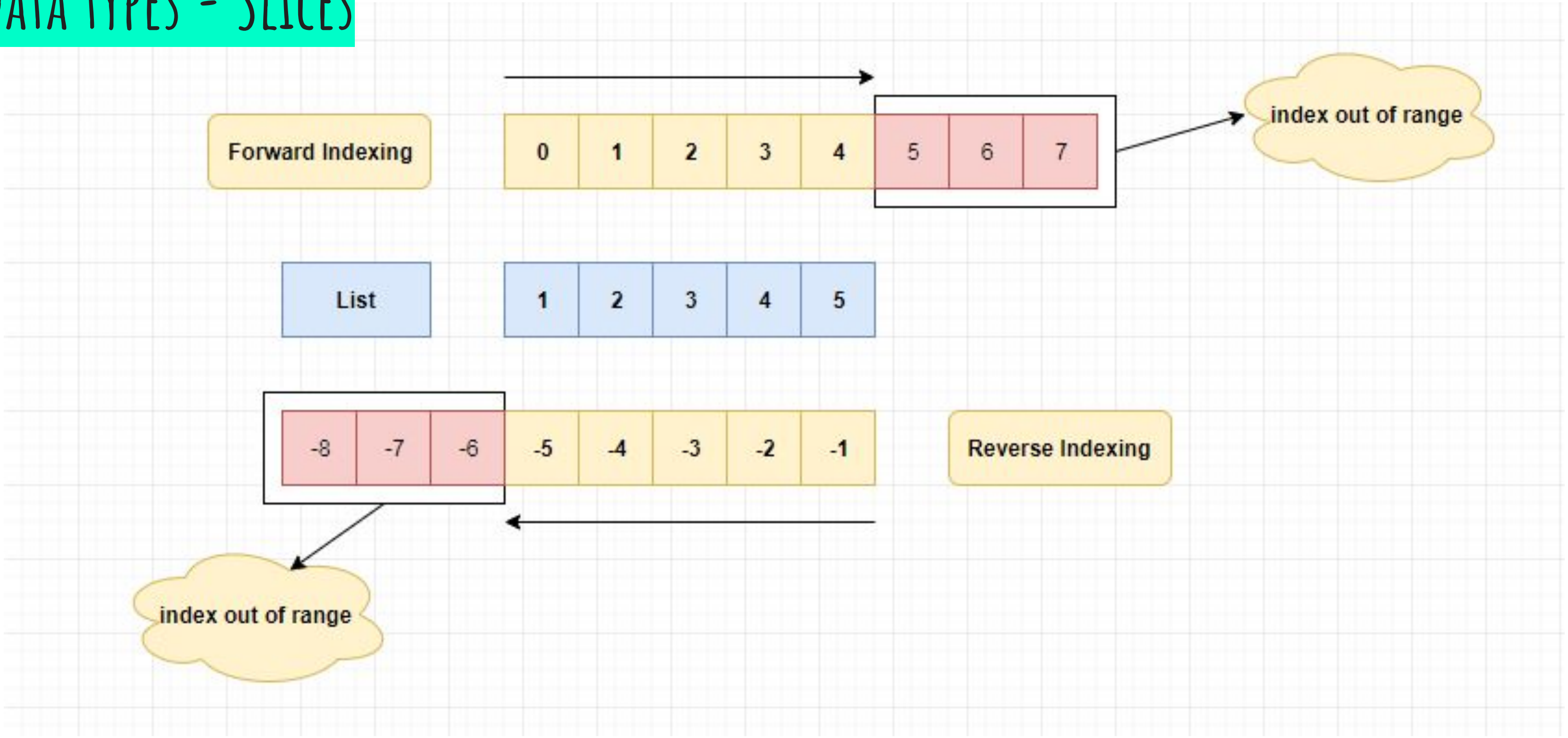
Name	Type	Description
Integers	int	Whole numbers, such as: **3 300 200**
Floating point	float	Numbers with a decimal point: **2.3 4.6 100.0**
Strings	str	Ordered sequence of characters: **"hello" 'Sammy' "2000" "楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

Data types practice

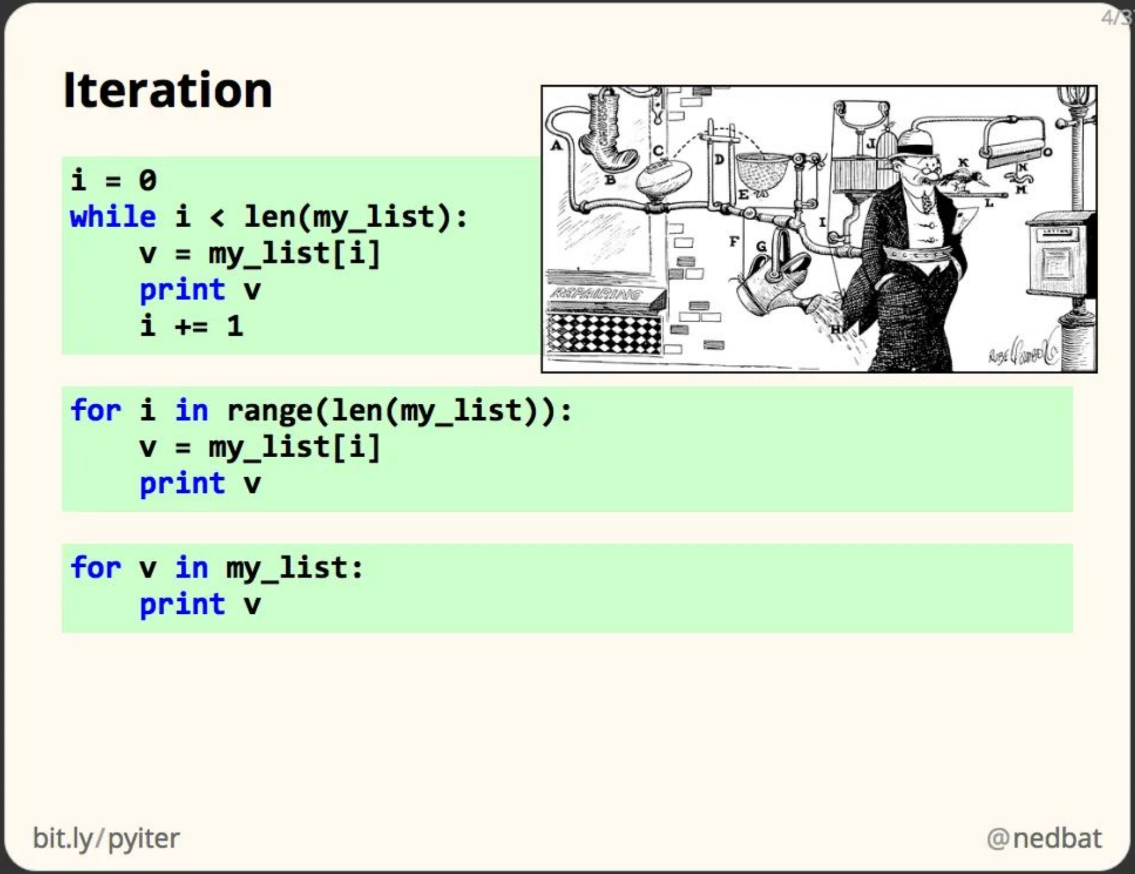


```
str = 'AppDividend'
print(type(str))

int = 123
print(type(int))

float = 21.19
print(type(float))

negative = -19
print(type(negative))

dictionary = {'blog':'AppDividend'}
print(type(dictionary))

list = [1, 2, 3]
print(type(list))

tuple = (19, 21, 46)
print(type(tuple))
```

Data types - Slices



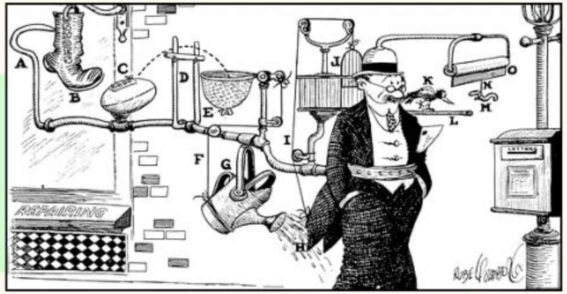
Iterations



4/37

Iteration

```
i = 0
while i < len(my_list):
    v = my_list[i]
    print v
    i += 1
```



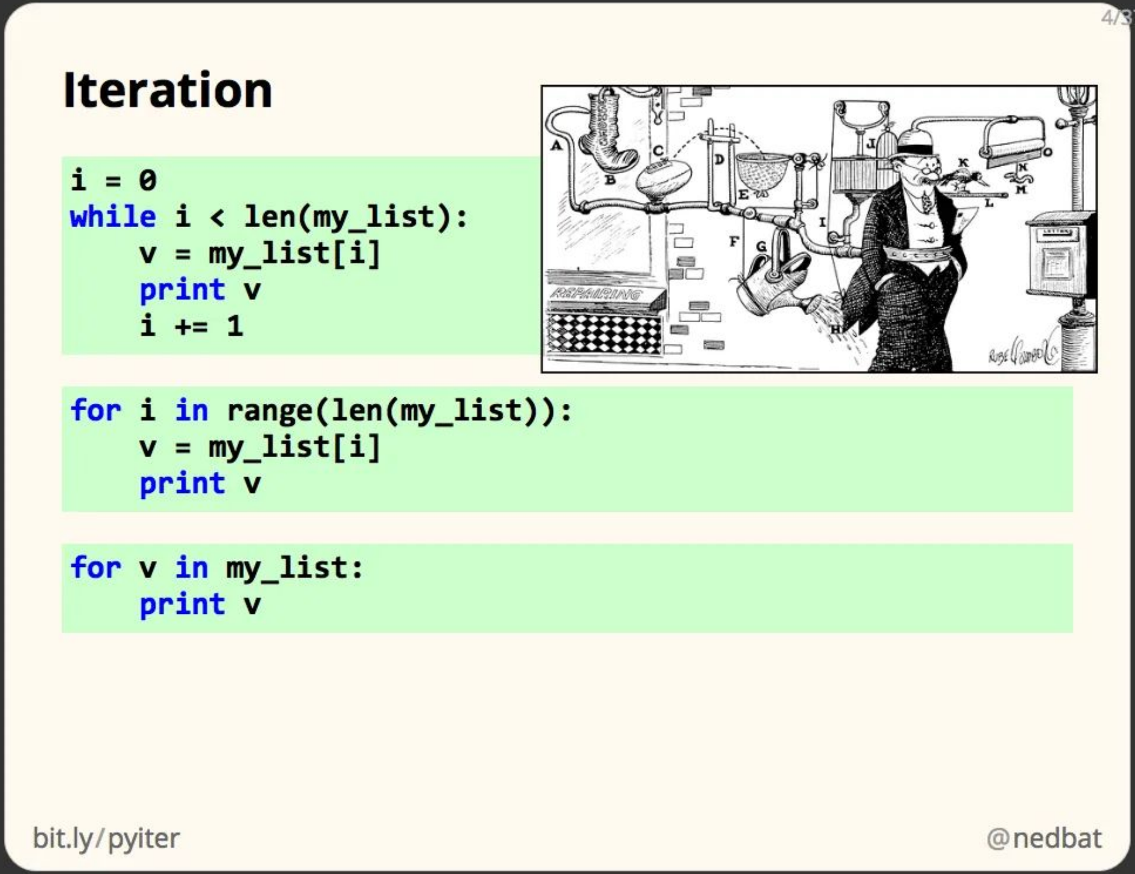
```
for i in range(len(my_list)):
    v = my_list[i]
    print v
```

```
for v in my_list:
    print v
```

bit.ly/pyiter

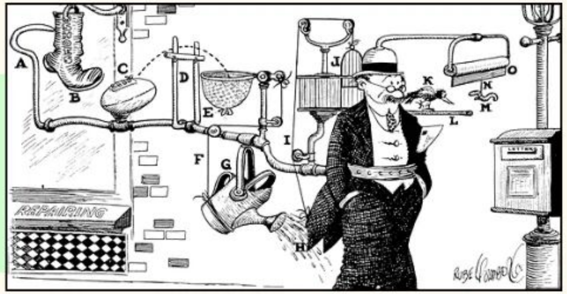
@nedbat

Iterations



Iteration

```
i = 0
while i < len(my_list):
    v = my_list[i]
    print v
    i += 1
```

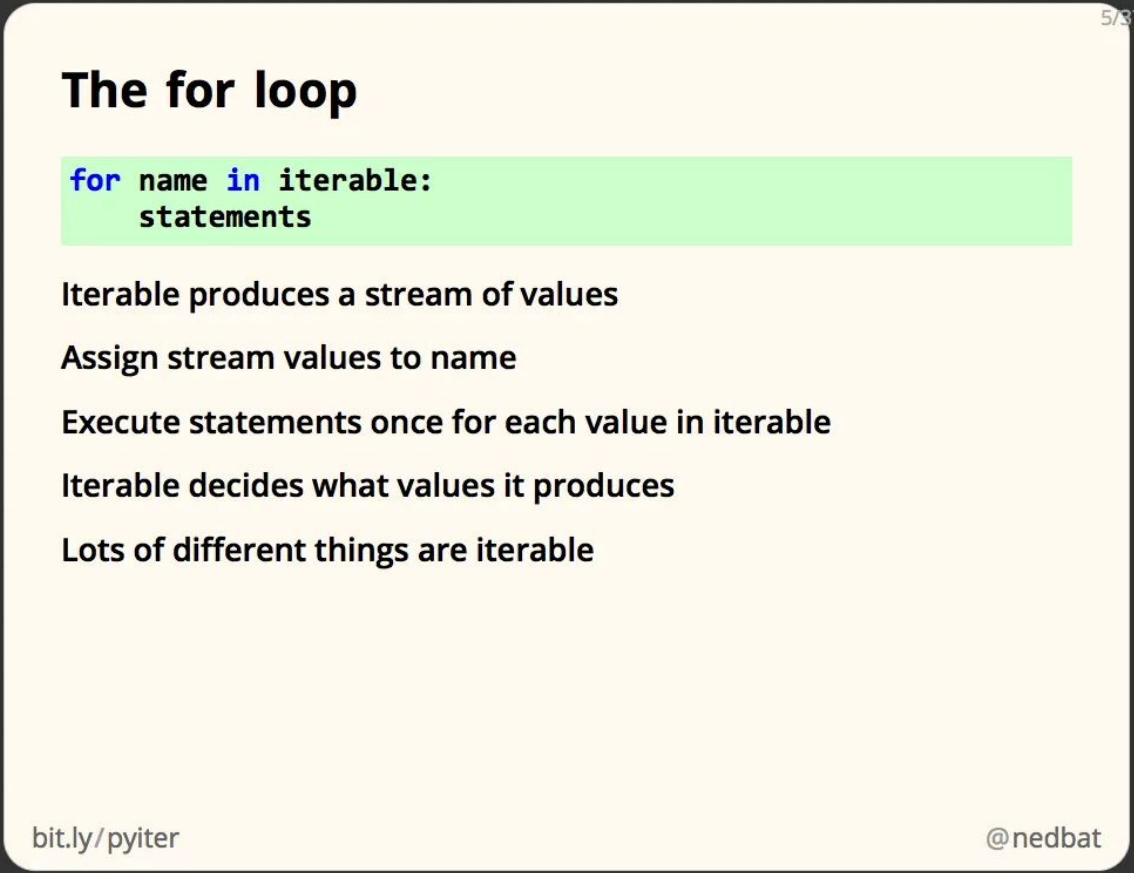


```
for i in range(len(my_list)):
    v = my_list[i]
    print v
```

```
for v in my_list:
    print v
```

bit.ly/pyiter @nedbat

Iterations



The for loop

```
for name in iterable:
    statements
```

Iterable produces a stream of values

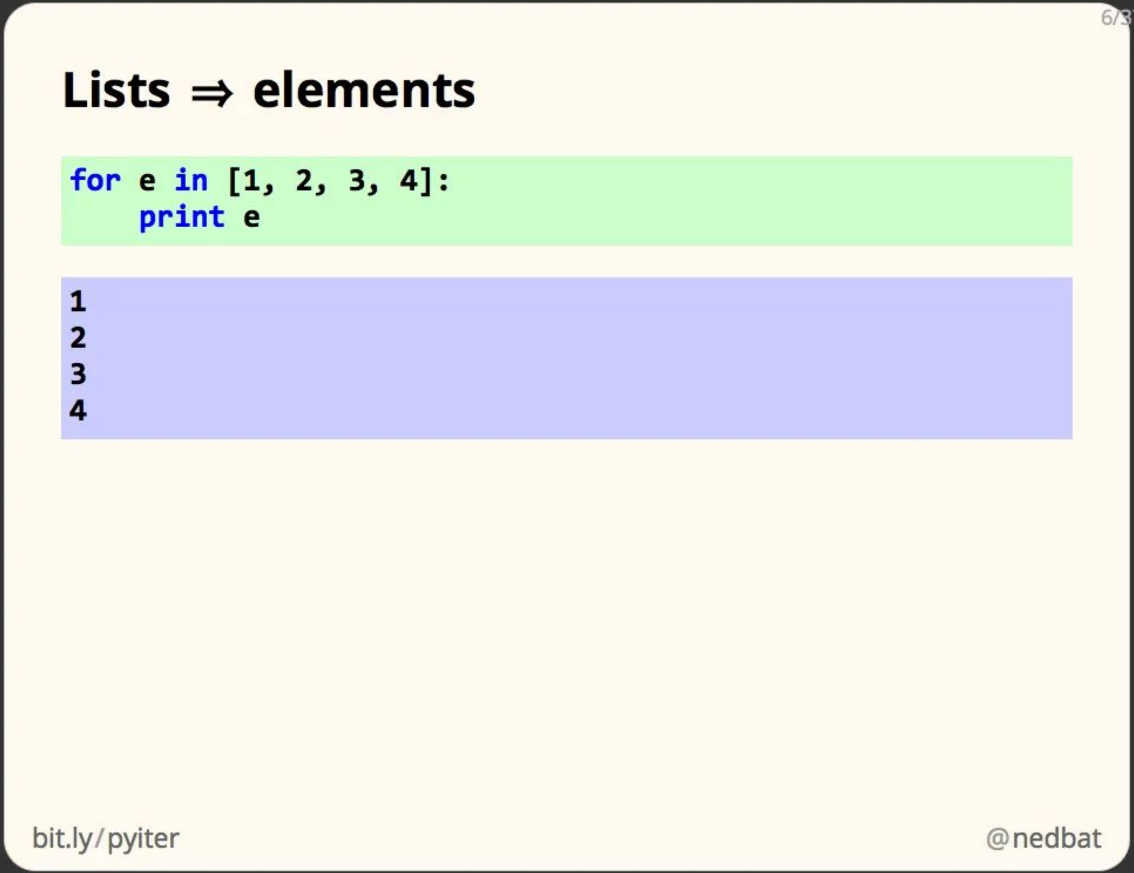
Assign stream values to name

Execute statements once for each value in iterable

Iterable decides what values it produces

Lots of different things are iterable

Iterations



Lists ⇒ elements

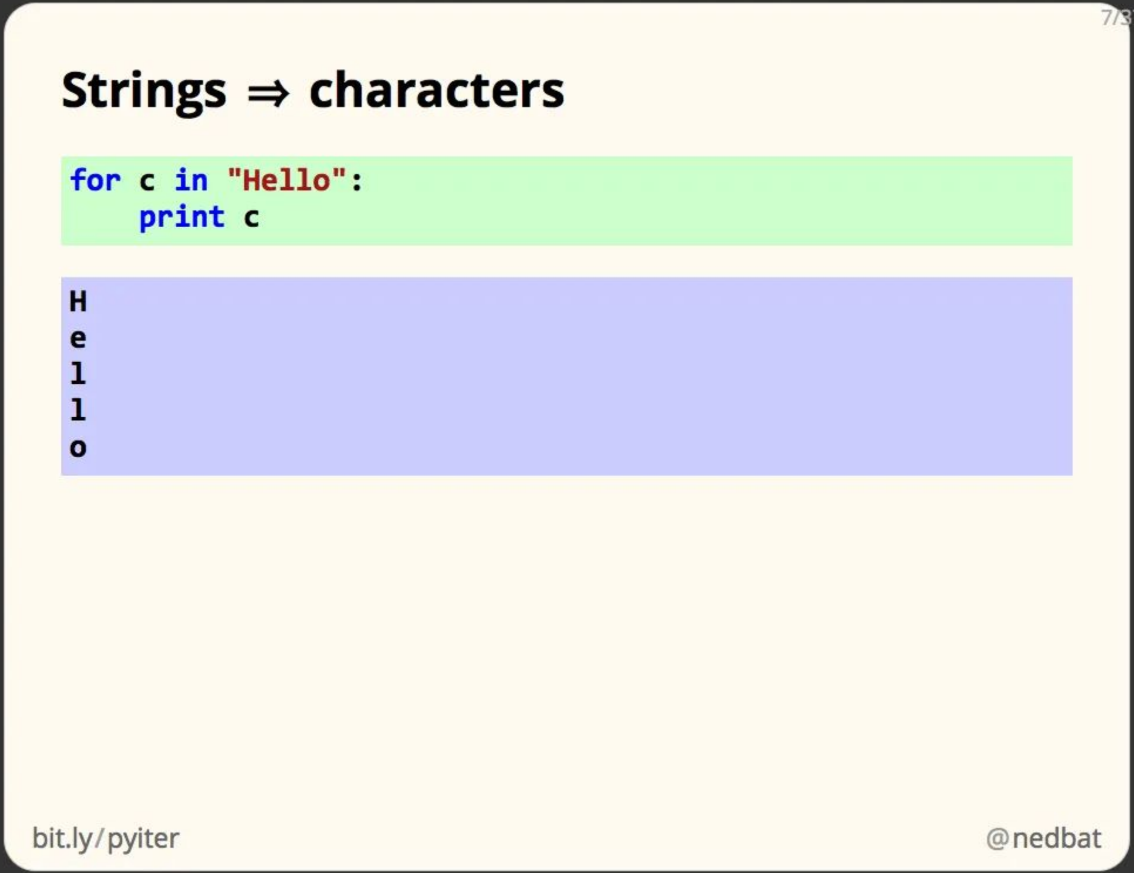
```python
for e in [1, 2, 3, 4]:
    print e
```

```
1
2
3
4
```

bit.ly/pyiter

@nedbat

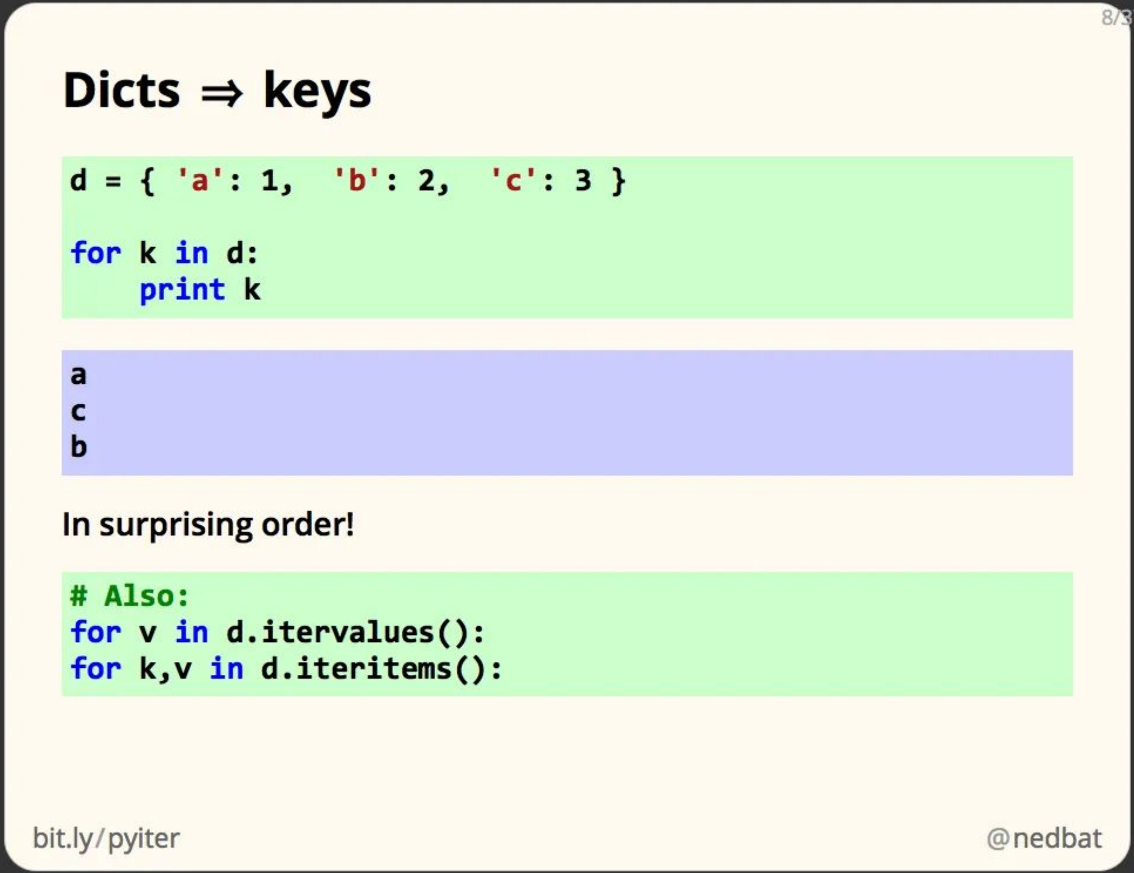
Iterations



Strings ⇒ characters

```
for c in "Hello":
    print c
```

```
H
e
l
l
o
```



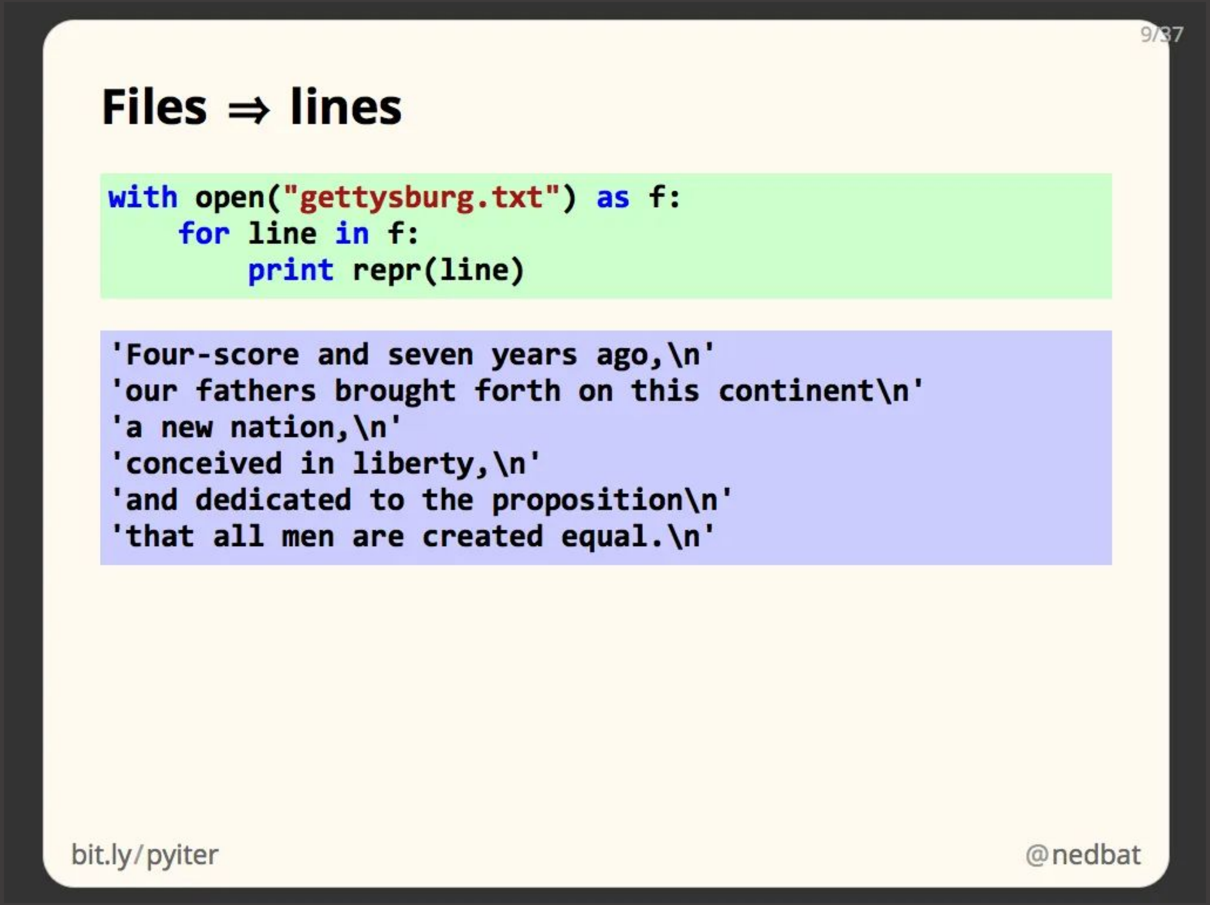
Dicts ⇒ keys

```
d = { 'a': 1,  'b': 2,  'c': 3 }

for k in d:
    print k
```

```
a
c
b
```

In surprising order!

```
# Also:
for v in d.itervalues():
for k,v in d.iteritems():
```

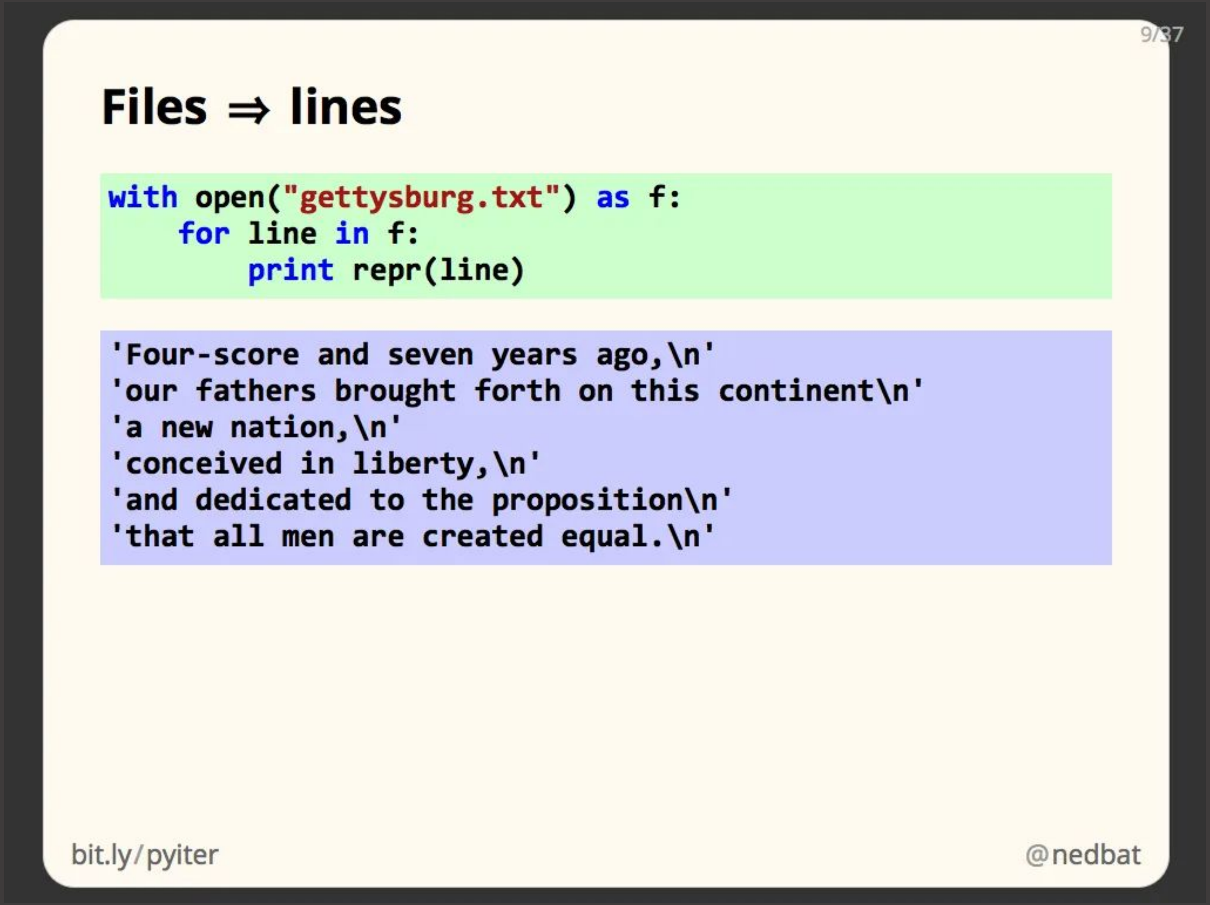


Files ⇒ lines

```
with open("gettysburg.txt") as f:
    for line in f:
        print repr(line)
```

```
'Four-score and seven years ago,\n'
'our fathers brought forth on this continent\n'
'a new nation,\n'
'conceived in liberty,\n'
'and dedicated to the proposition\n'
'that all men are created equal.\n'
```

Iterations



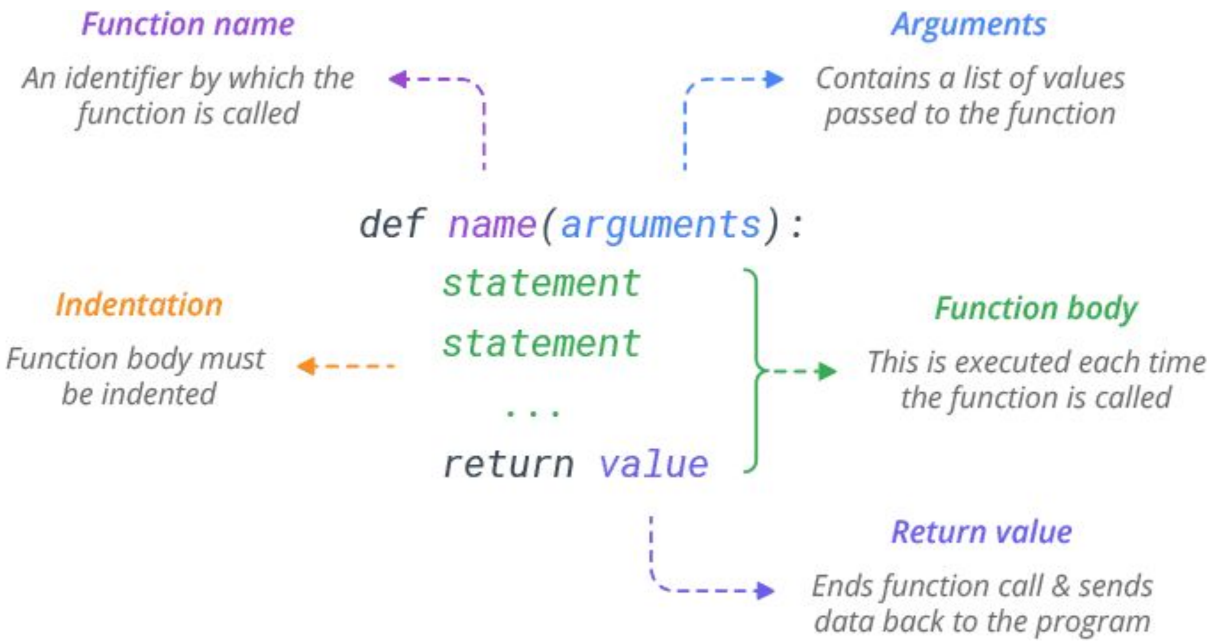
Files ⇒ lines

```
with open("gettysburg.txt") as f:
    for line in f:
        print repr(line)
```

```
'Four-score and seven years ago,\n'
'our fathers brought forth on this continent\n'
'a new nation,\n'
'conceived in liberty,\n'
'and dedicated to the proposition\n'
'that all men are created equal.\n'
```

Functions

Functions - recap


Function name
An identifier by which the function is called
Arguments
Contains a list of values passed to the function
def name(arguments):
statement
statement
...
return value
Indentation
Function body must be indented
Function body
This is executed each time the function is called
Return value
Ends function call & sends data back to the program

Functions - Template

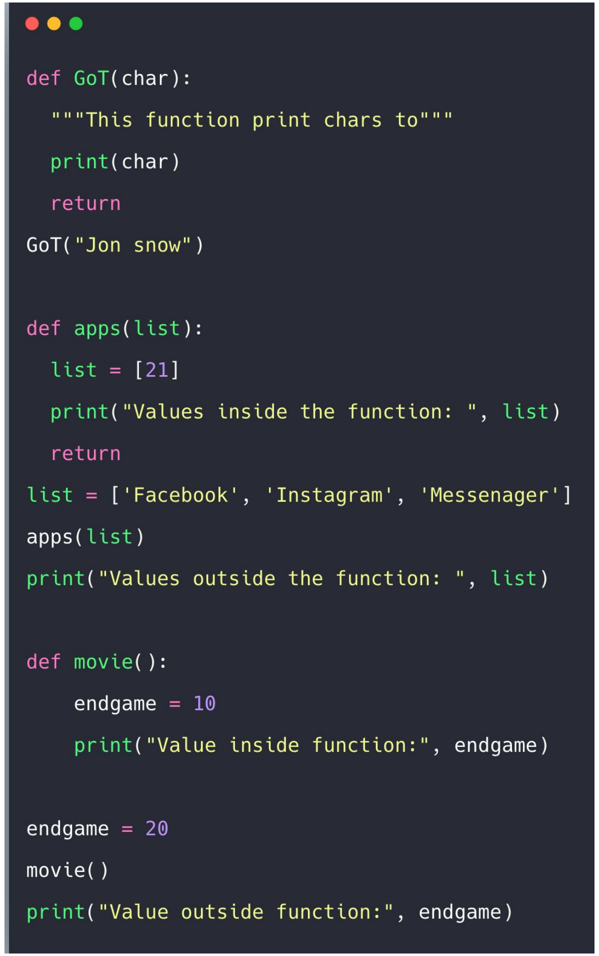
```
def name_of_the_function(arguments):
    instructions
    return result # optional: allows to reuse the result of a calculation outside the function
```

Functions - Example

```
def name_of_the_function(arguments):
    instructions
    return result # optional: allows to reuse the result of a calculation outside the function
```

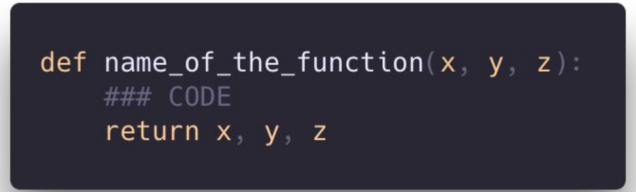
```
# Declaration of the function
def square_number(number):
    result = number**2
    print('The square of ', number, ' is ', result)

# Calls to the function: The code contained in the function is executed in this step.
square_number(3)
square_number(4)
square_number(12)
```

Functions - Example



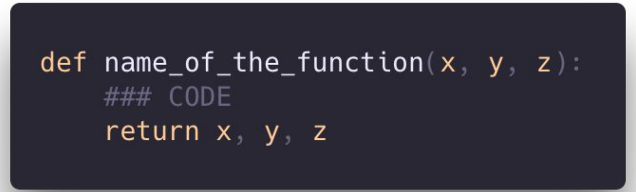
```
def GoT(char):
  """This function print chars to"""
  print(char)
  return
GoT("Jon snow")

def apps(list):
  list = [21]
  print("Values inside the function: ", list)
  return
list = ['Facebook', 'Instagram', 'Messenager']
apps(list)
print("Values outside the function: ", list)

def movie():
    endgame = 10
    print("Value inside function:", endgame)

endgame = 20
movie()
print("Value outside function:", endgame)
```

Functions - Multi-arguments - Template

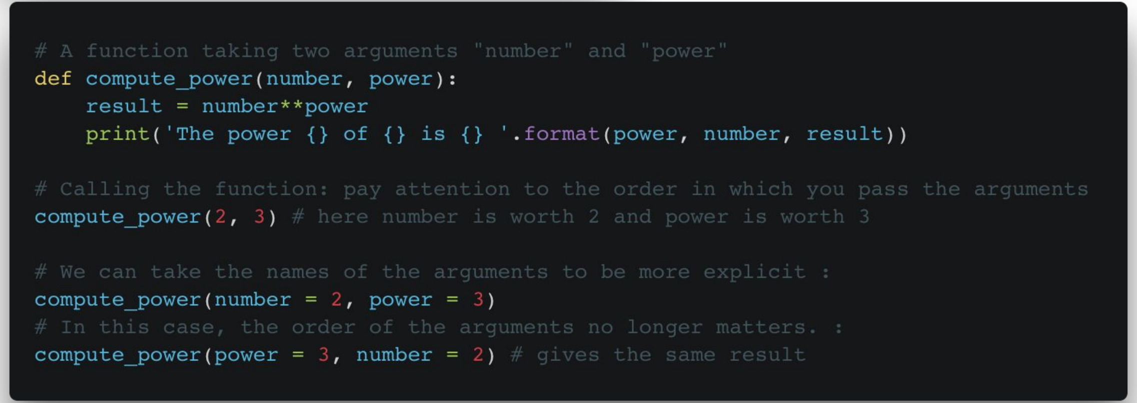


```
def name_of_the_function(x, y, z):
    ### CODE
    return x, y, z
```

Functions - Multi-arguments - Template

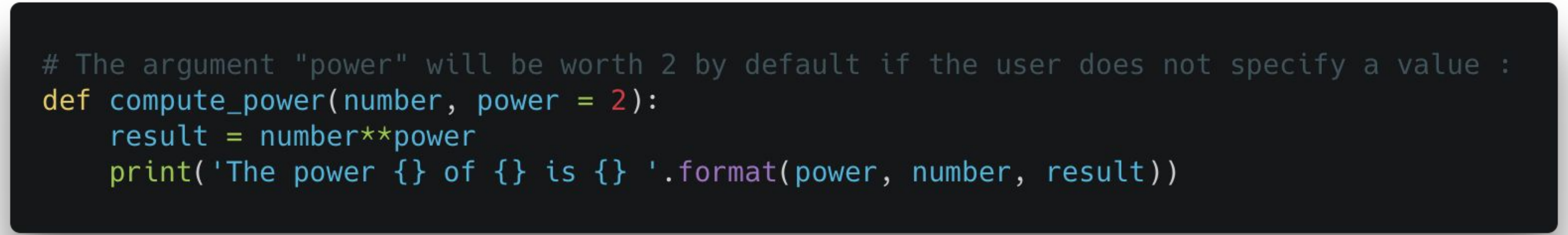


```python
def name_of_the_function(x, y, z):
    ### CODE
    return x, y, z
```



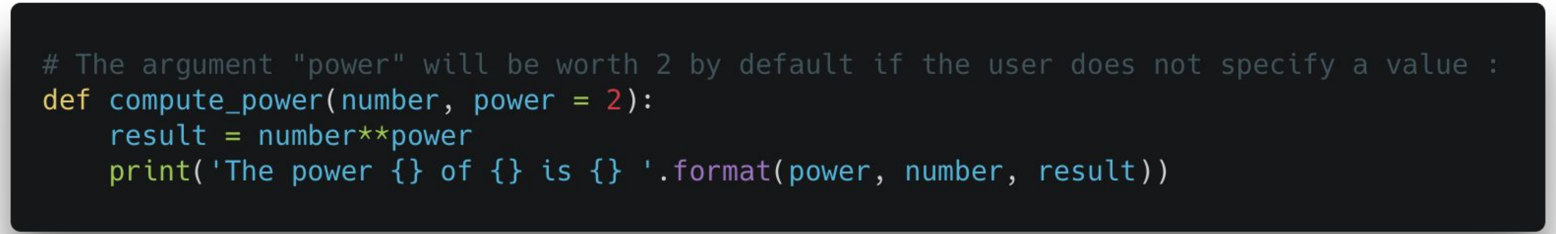
```python
# A function taking two arguments "number" and "power"
def compute_power(number, power):
    result = number**power
    print('The power {} of {} is {} '.format(power, number, result))

# Calling the function: pay attention to the order in which you pass the arguments
compute_power(2, 3) # here number is worth 2 and power is worth 3

# We can take the names of the arguments to be more explicit :
compute_power(number = 2, power = 3)
# In this case, the order of the arguments no longer matters. :
compute_power(power = 3, number = 2) # gives the same result
```

Functions - with default argument

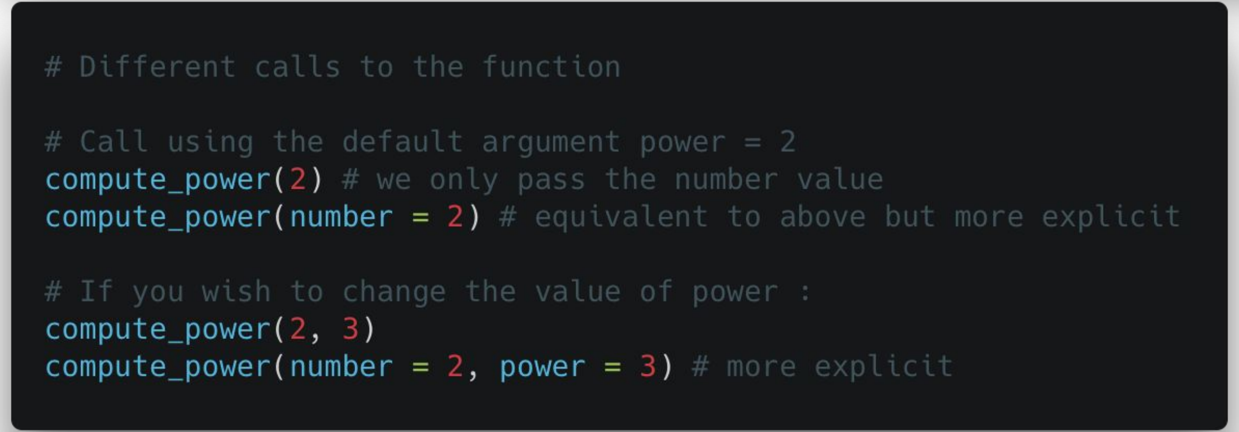


```python
# The argument "power" will be worth 2 by default if the user does not specify a value :
def compute_power(number, power = 2):
    result = number**power
    print('The power {} of {} is {} '.format(power, number, result))
```

Functions - with default argument

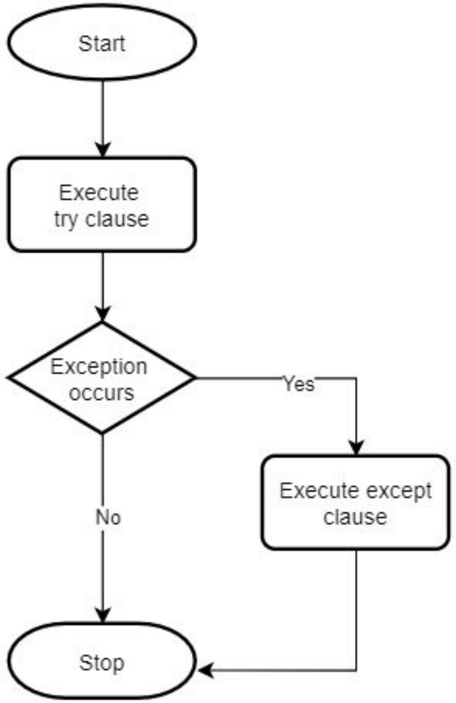


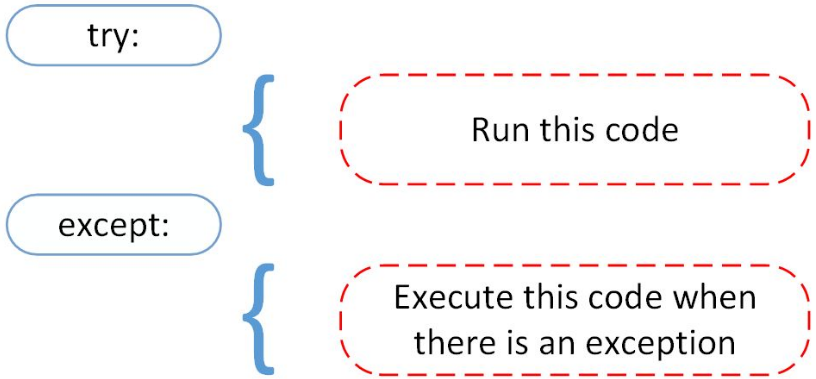
```python
# The argument "power" will be worth 2 by default if the user does not specify a value :
def compute_power(number, power = 2):
    result = number**power
    print('The power {} of {} is {} '.format(power, number, result))
```



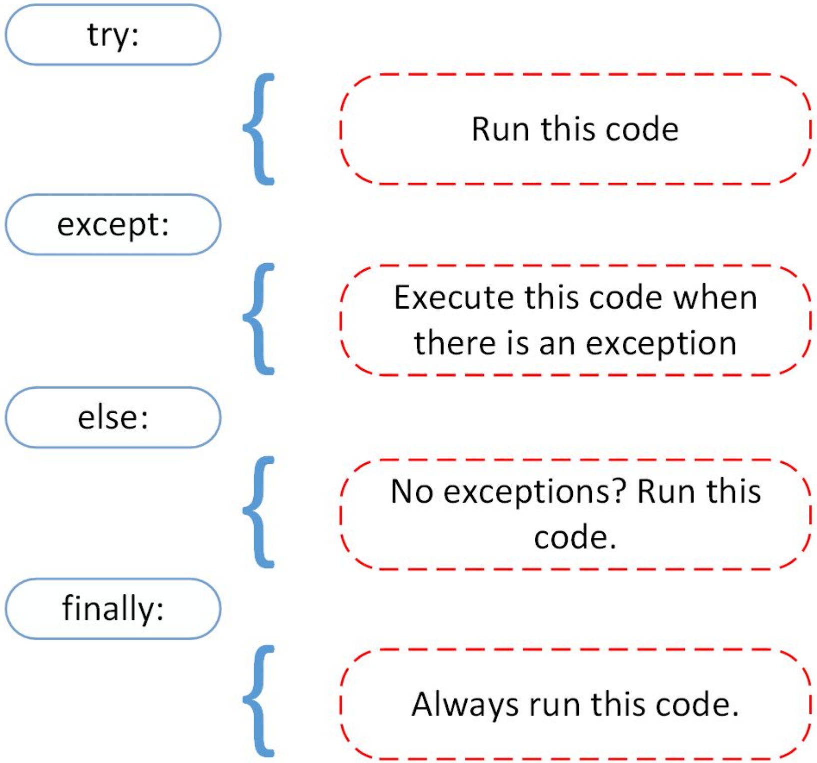
```python
# Different calls to the function

# Call using the default argument power = 2
compute_power(2) # we only pass the number value
compute_power(number = 2) # equivalent to above but more explicit

# If you wish to change the value of power :
compute_power(2, 3)
compute_power(number = 2, power = 3) # more explicit
```

Manage Exceptions





Manage Exceptions


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.
finally:
Always run this code.