|Sustainable water management

Python for Hydrology

Session 2 – Python

Objective:

Understand the way in which Python behaves, and explore all the types of tools that Python has as default

Lexical analysis

It is important to remember that we are using Python 3, Python 2 is a little bit different; moreover, it is deprecated.

Start opening Jupyter Lab and creating a new folder called "Session2", inside it add a new notebook of the same name.

To write code in Python, you have to write it bylines. The end of a logical line is the end of the line itself. Let's start commenting some text in Python, which is not going to be read by Python. It is used mainly as an instruction or reference to what the code means. To do this, use the following command:

```
#Session2 Python
```

```
[1]: #Session2 Python
```

If you type some random text inside Python you get an error because it is not defined

```
[3]: Sesssion2

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-3b906b892c74> in <module>
----> 1 Sesssion2

NameError: name 'Sesssion2' is not defined
```

Unless you type a number, Python identifies it automatically

```
[4]: 1
```

```
[4]: 1
```

As we move forward in the course, we would get deeper in the Python syntax, but let's start defining the data we can input to Python.

Numerical Types

Let's start with the numbers. The two types of numbers are integers and floats.

And integer has no decimals

A simple integer would be

```
8
```

```
[9]: 8

[9]: 8
```

Or a 0

```
0
```

```
[10]: 0

[10]: 0
```

But you can not declare a number with a cero at the start, this returns a Syntax error.

```
08
```

```
[11]: 08

  File "<ipython-input-11-c7e80dc713c2>", line 1
    08
     ^
SyntaxError: leading zeros in decimal integer literals are not permitted; use an 0o prefix for octal integers
```

You can specify a negative number

```
-123
```

```
-123
```

```
-123
```

You can also declare a number separated by underscores

```
1_2_3
```

```
1_2_3
```

```
123
```

You can make operations between the numbers like a sum:

```
5+8
```

```
5 + 8
```

```
13
```

A subtraction

```
90 - 10
```

```
90 - 10
```

```
80
```

A multiplication

```
4*7
```

```
4 * 7
```

```
28
```

A division, which returns us a float

```
8/2
```

```
8 / 2
```

```
4.0
```

We can get the quotient

```
7 // 2
```

```
7 // 2
```

```
3
```

For the remainder

```
7%3
```

```
7 % 3
```

```
1
```

For the exponential

```
3**4
```

```
3**4
```

```
81
```

We can do multiple operations in one line

```
5 + 8 - 9 * 5
```

```
5 + 8 - 9 * 5
```

```
-32
```

An easy way to get the type of the variables is using the command "type", this like a function, we are going to talk about functions in advance.

```
type(4)
```

```
int
```

You can declare float numbers just by adding the decimal point

```
4.
```

```
4.
```

```
4.0
```

You can add all the decimals you want

```
4.595949484188
```

```
4.595949484188
```

```
4.595949484188
```

If you make an operation with an integer and a float, you automatically get a float number

```
4.*2
```

```
4.*2
```

```
8.0
```

Additionally, you can convert a float to an integer

```
int(5.)
```

```
int(5.)
```

```
5
```

The integer returned in the previous step can be transformed to a float.

```
float(int(5.))
```

```
float(int(5.))
```

```
5.0
```

One of the most useful utilities that a programming language has is to declare variables. You can call the variable in whatever way you want, but be sure not to declare the variable name with a number as the first character so that you can do the following

```
a =9
a
```

```
a =9
a
```

```
9
```

If the first character of the variable is a number, you automatically get an error

```
0a = 0
0a
```

```
0a = 0
0a
```

```
  File "<ipython-input-1-3fa6528
    0a = 0
     ^
SyntaxError: invalid syntax
```

A variable can contain a multiplication and a sum

```
a = 9 + 18 - 20
a
```

```
a = 9 + 18 - 20
a
```

```
7
```

You make operations with the variable.

```
a -3
```

```
a -3
```

```
4
```

Doing the previous operation does not change the value of "a"

```
a
```

```
7
```

If you want to change the value of "a" like, for example, adding it 5

```
a = a +5
a
```

```
a = a +5
a
```

```
12
```

A quicker way to do the previous addition is of the following way:

```
a += 5
a
```

```
a += 5
a
```

```
12
```

A sum can have a breakline by using "\".

```
sum = 0
sum += 1 +\
     5 + \
     8 + \
     7
sum
```

```
sum = 0
sum += 1 +\
    5 + \
    8 + \
    7
sum
```

```
21
```

An example of numerical operation using variables could be converting from Celcius degrees to Kelvin

```
temp_C = 25 #°C
temp_K = 25 + 273 #K
temp_K
```

```
temp_C = 25 #°C
temp_K = 25 + 273 #K
temp_K
```

```
298
```

Let's introduce to Boolean values. they can be "True" or "False."

```
True, False
```

```
True, False
```

```
(True, False)
```

Boolean values can be related to integers, "True" is related to 1, and "False" is related to 0, so you can make sums with them.

```
False + 1 , True +1
```

```
False + 1 , True +1
```

```
(1, 2)
```

Strings

A string is a text, the way you add it is enclosing it by "":

```
"Texto"
```

```
"Texto"
```

```
'Texto'
```

You can also use ` ’ to define text

```
'Texto'
```

```
'Texto'
```

```
'Texto'
```

If you want to have a text with quotes, you need to use the combination of the previous quotes defined

```
"Texto 'texto'"
```

```
"Texto 'texto'"
```

```
"Texto 'texto'"
```

If you want to concatenate text, the easiest way to do this is by making a sum.

```
"Texto" + "Texto2"
```

```
"Texto" + "Texto2"
```

```
'TextoTexto2'
```

A multi-line text is defined with triple quotes, if you check the output, you could see the keywords "\n", this keyword means a linebreak, this keyword is read in the kernel

```
'''
Welcome to the course,
we are in the session 2
'''
```

```
'''
Welcome to the course,
we are in the session 2
'''
```

```
'\nWelcome to the course,\nwe are in the session 2\n'
```

If you want to format a number inside a text, you can do use the "%" keyword like in the following example

```
number = 1
"%s"%number
```

```
number = 1
"%s"%number
```

```
'1'
```

The previous way of inserting text is useful when you are writing text, and you want to pass variables to it.

```
('This one is like converting to a string %s, this one like a integer
%i, this one like a float %f, and this one is a scientific notation
%e')%(number, number, number, number)
```

```
('This one is like converting to a string %s, this one like a integer %i, this one like a float %f, and this one is a scientific notation %e')%(number, n
```

```
'This one is like converting to a string 1, this one like a integer 1, this one like a float 1.000000, and this one is a scientific notation 1.000000e+0
0'
```

A new way to do the previous example is using "F-strings", to use it, you need to write a "F" before the quotes, and the value must be declared inside brackets

```
F"This example is using F-string {number}"
```

```
F"This example is using F-string {number}"
```

```
'This example is using F-string 1'
```

In the same way, a "F-string" can be formatted.

```
F"This example is using F-string {number:f}"
```

```
F"This example is using F-string {number:f}"
```

```
'This example is using F-string 1.000000'
```

Each character of a string can be selected, Python uses the 0 as the first value, so if we want to get first value of a string we use the following script:

```
texto = '1234'
texto[0]
```

```
texto = '1234'
texto[0]
```

```
'1'
```

To get the second value:

```
texto[1]
```

```
texto[1]
```

```
'2'
```

You can find the values starting from the last character using the negative sign

```
texto[-1]
```

```
texto[-1]
```

```
'4'
```

To get the first value:

```
texto[-4]
```

```
texto[-4]
```

```
'1'
```

You can form new strings by slicing

```
texto[-4]+ texto[-1]
```

```
texto[-4]+ texto[-1]
```

```
'14'
```

A useful tool to know the length of characters is using “len”

```
len(t)
```

```
len(t)
```

```
5
```

You can replace values using "replace"

```
text = '                    Welcome to session 1                    '
text.replace('1' , '2')
```

```
text = '                    Welcome to session 1                    '
text.replace('1' , '2')
```

```
'                    Welcome to session 2                    '
```

If you want to delete the extra spaces in the left and right you can use "strip"

```
text.strip()
```

```
text.strip()
```

```
'Welcome to session 1'
```

You can identify if a text starts with some text using "startswith"

```
text.startswith('Welcome')
```

```
text.startswith('Welcome')
```

```
True
```

Or use "endswith" to know if the text ends with some defined character

```
text.endswith('1')
```

```
text.endswith('1')
```

```
False
```

If you want to know the position of a character, type the following:

```
text.find('1')
```

```
text.find('1')
```

```
19
```

If you want to upper case the first character, use "capitalize"

```
text.capitalize()
```

```
text.capitalize()
```

```
'                    welcome to session 1                    '
```

To format it as a title, use "title"

```
text.title()
```

```
text.title()
```

```
'                    Welcome To Session 1                    '
```

If you want to upper case the entire text use "upper"

```
text.upper()
```

```
text.upper()
```

```
'                    WELCOME TO SESSION 1                    '
```

We can make more interesting things like the following:

```
float(text[text.find('1')])
```

```
float(text[text.find('1')])
```

```
1.0
```

As we saw, if you write some text, the kernel automatically returns the text, but this does not happen when you are in the kernel. If you want to return some text, you need to use the "print" function.

```
print("sumatoria")
```

```
print("sumatoria")
```

```
sumatoria
```

Another example:

```
print('Welcome','to','session','2')
```

```
print('Welcome','to','session','2')
```

```
Welcome to session 2
```

If - conditional

If-conditionals are useful when you want to return values base on some input

For example, we can define a variable with the value "True", and as it is True, it returns the argument of the condition

```
sesion2 = True
if sesion2:
    "We are in session 2"
else:
    "We are not in session 2"
```

```
sesion2 = True
if sesion2:
    "We are in session 2"
else:
    "We are not in session 2"
```

The previous script returns nothing, so, as mentioned before, you need to declare the text with a "print"

```
sesion2 = True
if sesion2:
    print("We are in session 2")
else:
    print("We are not in session 2")
```

```
sesion2 = True
if sesion2:
    print("We are in session 2")
else:
    print("We are not in session 2")
```

```
We are in session 2
```

If we change the value to "False" it passes through the "If" condition and goes to the "else" condition.

```
sesion2 = False
```

```
if sesion2:
    print("We are in session 2")
else:
    print("We are not in session 2")
```

```
sesion2 = False
if sesion2:
    print("We are in session 2")
else:
    print("We are not in session 2")
```

Let's introduce the comparison signs, which always returns a Boolean.

A greater than symbol example is as follows:

```
x=4
x > 5
```

```
x=4
x > 5
```

```
False
```

An equality sign is as follows:

```
x == 4
```

```
x == 4
```

```
True
```

You can have multiple comparison signs.

```
2 < x < 10
```

```
2 < x < 10
```

```
True
```

Another way of using them

```
( 5 > x ) and ( 10 > x)
```

```
( 5 > x ) and ( 10 > x)
```

```
True
```

If we want a negative condition, we could use the following example:

```
5 > x and not 10 < x
```

```
5 > x and not 10 < x
```

```
True
```

We can use these conditionals to create an If-condition

```
x = 4
if x < 4:
    print("The number is less than one")
else:
    print("We are not in session 2")
```

```
x = 4
if x < 4:
    print("The number is less than one")
else:
    print("We are not in session 2")
```

```
The number is less than one
```

Lists

Lists are a way to lump variables or data together; to create one, you need to declare all the data into parentheses

```
vowel = ['a', 'b', 'c', 'd', 'e']
vowel
```

```
vowel = ['a', 'b', 'c', 'd', 'e']
vowel
```

```
['a', 'b', 'c', 'd', 'e']
```

You can wrap any kind of information you want.

```
['a','1','b',580]
```

```
['a','1','b',580]
```

```
['a', '1', 'b', 580]
```

You can have nested lists

```
['a','1','b',580 , ['b','c',['nested']]]
```

```
['a','1','b',580 , ['b','c',['nested']]]
```

```
['a', '1', 'b', 580, ['b', 'c', ['nested']]]
```

As we slice data with the strings we can slice data too

```
lista = ['a','1','b',580]
lista[1]
```

```
list = ['a','1','b',580]
list[1]
```

```
'1'
```

We can sum list, which produces a concatenated one

```
list += ['2','q','nueva','lista']
list
```

```
list += ['2','q','nueva','lista']
list
```

```
[5, '1', 'b', 580, '2', 'q', 'nueva', 'lista']
```

Another way to slice is in a range. In the following example, we indicate to Python to slice from index 0 to 4, but Python does not consider the 4, only until one index before 4.

```
list[0:4]
```

```
list[0:4]
```

```
[5, '1', 'b', 580]
```

We can use negative indexes too, but be careful because it only slices from left to right

```
list[-1:4]
```

```
list[-1:4]
```

```
[]
```

The right way to do this using negative indexes is:

```
list[-5:-1]
```

```
list[-5:-1]
```

```
[580, '2', 'q', 'nueva']
```

If we write the ":" symbol without another number, it considers that the slicing is till the end of the list.

```
list[-5:]
```

```
[580, '2', 'q', 'nueva', 'lista']
```

But slicings can consider a third number related to the step

```
list[0:4:1]
```

```
list[0:4:1]
```

```
['a', '1', 'b', 580]
```

So if we consider a negative step, we can slice from right to left.

```
list[-1:-5:-1]
```

```
list[-1:-5:-1]
```

```
[580, 'b', '1', 'a']
```

To get the length of a list, we use "len"

```
len(list)
```

```
len(list)
```

```
4
```

If you want to remove a value from the list, use "remove"

```
list.remove(580)
list
```

```
list.remove(580)
list
```

```
[5, '1', 'b', '2', 'q', 'nueva', 'lista']
```

To add an item to the end of the list

```
list.append(580)
list
```

```
list.append(580)
list
```

```
[5, '1', 'b', '2', 'q', 'nueva', 'lista', 580]
```

You can drop the last value of the list using "pop",

```
list.pop()
```

```
list.pop()
```

```
580
```

You can drop values with a specified index too.

```
list.pop(2)
```

```
list
```

```
list.pop(2)
list
```

```
[5, '1', '2', 'q', 'nueva', 'lista']
```

If you want to check if a value is in a list:

```
5 in list
```

```
5 in list
```

```
True
```

You can assign a variable based on a list:

```
listb = list
listb
```

```
listb = list
listb
```

```
[5, '1', '2', 'q', 'nueva', 'lista']
```

But beware, because if you change any value in one list, the other list is affected.

```
listb[0] = 6
listb[0],list[0]
```

```
listb[0] = 6
listb[0],list[0]
```

```
(6, 6)
```

If you want to create an independent copy, you need to use "copy"

```
list2 = list.copy()
list2
```

```
list2 = list.copy()
list2
```

```
[6, '1', '2', 'q', 'nueva', 'lista']
```

And the former list is not affected if you make changes in the copied one.

```
list2[0] = 1000
list2[0],list[0]
```

```
list2[0] = 1000
list2[0],list[0]
```

```
(1000, 6)
```

Tuples are another way of lumping together data, but they are immutable what does it mean is that they cannot be modified once created.

```
vowel = ('a', 'b', 'c', 'd', 'e')
vowel
```

```
vowel = ('a', 'b', 'c', 'd', 'e')
vowel
```

```
('a', 'b', 'c', 'd', 'e')
```

You can get data by slicing, but you cannot modify its values.

```
vowel[1] = 11
```

```
vowel[1] = 11
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-b6d9e430d480> in <module>
----> 1 vowel[1] = 11

TypeError: 'tuple' object does not support item assignment
```

Loop with while and for

Let's start with the "while" loops. A while statement loops until a condition is met. You need to be extremely careful because if you don't add a condition, Python iterates until your computer crashes. Lucky you, Jupyter Lab has a Stop button that can help you in this situation.

A simple loop has the following shape:

```
a = 1
while a <10:
    print(a)
    a += 1
```

```
: a = 1
  while a <10:
      print(a)
      a += 1

  1
  2
  3
  4
  5
  6
  7
  8
  9
```

You can nest if conditions inside a "while" loop

```
a = 1
while a:
    print(a)
    if a ==10:
        break
    a += 1
```

```
a = 1
while a:
    print(a)
    if a ==10:
        break
    a += 1
```

```
a
```

```
10
```

For loops iterate over a range of given values, let's use the list created before to do this.

```
for i in list:
    print(i)
```

```
for i in list:
    print(i)
```

```
6
1
2
q
nueva
lista
```

Another way of doing a iteration is as follows:

```
for letter in 'abcdefghi':
    if letter == 'h':
        break
    else:
        print(letter)
```

```
for letter in 'abcdefghi':
    if letter == 'h':
        break
    else:
        print(letter)
```

```
a
b
c
d
e
f
g
```

You can create nested iterations too:

```
for i in ['a','b','c']:
    for j in [1,2,3]:
        print(i,j)
```

```
for i in ['a','b','c']:
    for j in [1,2,3]:
        print(i,j)
```

```
a 1
a 2
a 3
b 1
b 2
b 3
c 1
c 2
c 3
```

Dictionaries

To create a dictionary, you need to declare it with brackets

```
empty = {}
empty
```

```
empty = {}
empty
```

```
{}
```

A dictionary has two components the key and the value related to the key, an example of a one key one value dictionary is as follows:

```
dict = {'key':'value'}
dict
```

```
dict = {'key':'value'}
dict
```

```
{'key': 'value'}
```

A multi-key dictionary example:

```
user = {'name':'Jonathan','last name':'Quiroz','age':24}
user
```

```
user = {'name':'Jonathan','last name':'Quiroz','age':24}
user
```

```
{'name': 'Jonathan', 'last name': 'Quiroz', 'age': 24}
```

To get the keys of a dictionary

```
user.keys()
```

```
user.keys()
```

```
dict_keys(['name', 'last name', 'age'])
```

To get the values of the dictionary:

```
user.values()
```

```
user.values()
```

```
dict_values(['Jonathan', 'Quiroz', 24])
```

You can get a list with tuples of the keys and values

```
user.items()
```

```
user.items()
```

```
dict_items([('name', 'Jonathan'), ('last name', 'Quiroz'), ('age', 24)])
```

To get the length of the dict

```
len(user)
```

```
len(user)
```

```
3
```

You can get the values of the dictionaries by "slicing" the dictionary

```
user['name']
```

```
user['name']
```

```
'Jonathan'
```

If you want to delete a key, use "del"

```
del user['age']
user
```

```
del user['age']
user
```

```
{'name': 'Jonathan', 'last name': 'Quiroz'}
```

To empty all the dictionary, use "clear"

```
user.clear()
user
```

```
user.clear()
user
```

```
{}
```

You can have any kind of format inside the dictionary values, even a dictionary can be inside another dictionary.

```
things = {'names':['a'',b']}
things
```

```
things = {'names':['a'',b']}
things
```

```
{'names': ['a,b']}
```

Sets

A set object is enclosed in brackets as dictionary does, but the difference is like it is like a list of unique keys, you can make the classical operations of sets with it.

```
empty_set = set()
empty_set
```

```
empty_set = set()
empty_set
```

```
set()
```

A set of numbers would be:

```
numbers = {1,2,3,5,6,9,8,7,4,2,65}
```

```
numbers = {1,2,3,5,6,9,8,7,4,2,65}
```

You can transform a string to a set

```
set('letters')
```

```
set('letters')
```

```
{'e', 'l', 'r', 's', 't'}
```

Let's do some operations, first create two sets

```
a = {1,2}
b = {2,3}
```

```
a = {1,2}
b = {2,3}
```

To make an intersection, you can use either the symbol "&" and "intersection"

```
a&b, a.intersection(b)
```

```
a&b, a.intersection(b)
```

```
({2}, {2})
```

To make a union, you can either use the symbol "|" and "union"

```
a | b , a.union(b)
```

```
a | b , a.union(b)
```

```
({1, 2, 3}, {1, 2, 3})
```

To make a substraction, you can either use "-" or "difference"

```
a - b, a.difference(b)
```

```
a - b, a.difference(b)
```

```
({1}, {1})
```

A symmetrical difference can be defined as "^" or "symmetrical_difference"

```
a ^ b, a.symmetric_difference(b)
```

```
a ^ b, a.symmetric_difference(b)
```

```
({1, 3}, {1, 3})
```

Functions

A function is useful for repetitive procedures, like if you want to make a complicate calculation once and only repeat it multiple times.

The parts of a function are as follows:

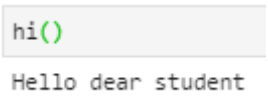
1. A "def" keyword declared at the start of the function
2. The name oif the function
3. The parameters of the function
4. The ":" sign that declares the end of the function
5. The statements
6. An optional "return" statement

An example of a simple function is as follows:

```
def hi():
    print('Hello dear student')
```

So, whenever you want to call this function, you just call it by its name:

```
hi()
```



An example of the usage of the return statement would be:

```
def hi():
    return "new student"
hi()
```

```
def hi():
    return "new student"
hi()
```

```
'new student'
```

The return statement is useful when you want to store the result of the function in a variable

```
a = hi()
a
```

```
a = hi()
a
```

```
'new student'
```

An example of a function with an argument is as follows:

```
#argument
def hi(your_name):
    print('Hello '+your_name)
```

```
#argument
def hi(your_name):
    print('Hello '+your_name)
```

If you call the function without the argument it returns an error indicating that you need to add the argument

```
hi()
```

```
hi()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-86-ce8dd9c0d491> in <module>
----> 1 hi()

TypeError: hi() missing 1 required positional argument: 'your_name'
```

So, you need to indicate the argument, if you know the order you can declare it directly

```
hi('Jonathan')
```

```
hi('Jonathan')
```

```
Hello Jonathan
```

You can also call the parameter key.

```
hi(your_name='Jonathan')
```

```
hi(your_name='Jonathan')
```

```
Hello Jonathan
```

Some functions have default values declared by default, an example of it is as follows:

```
def hi(your_name,your_lastname='Quiroz'):
    print(F"Hello {your_name} {your_lastname}")
```

```
def hi(your_name,your_lastname='Quiroz'):
    print(F"Hello {your_name} {your_lastname}")
```

In this case, as there is a parameter with a default value you can indicate the parameter without default value.

```
hi('Jonathan')
```

```
hi('Jonathan')
```

```
Hello Jonathan Quiroz
```

Or overwrite the default value if you want to.

```
hi('Jonathan', 'Quiros Valdivia')
```

```
hi('Jonathan', 'Quiros Valdivia')
```

```
Hello Jonathan Quiros Valdivia
```

A function can have all the features we have seen before, for example a function with a conditional:

```
def number(n):
    if n < 100:
        print('The number is lower than 100')
    else:
        print('The number is a greater than 100')
```

```
def number(n):
    if n < 100:
        print('The number is lower than 100')
    else:
        print('The number is a greater than 100')
```

Let's test it

```
number(50)
```

```
number(50)
```

```
The number is lower than 100
```

```
number(155)
```

```
number(155)
```

```
The number is a greater than 100
```

The trick with Python is that given all these tools, let your imagination solve problems, you can combine all the tools given by Python to make anything you want.

Reference Links:

- https://www.python.org/
- Any Python 3 books, the ones from O'REILLY are good and have a lot of series