Introduction to programming.

Lecturers : Giovanni Casini (giovanni.casini@uni.lu), Xavier Parent (xavier.parent@uni.lu) The slides and the handout have been obtained modifying the materials by Clément Guérin

In the process of solving a problem with a program, you have some general steps :

- Understanding the problem.
- Converting the abstract problem into a computer-compatible process, i.e. write the algorithm solving the problem.
- Converting the algorithm in a valid script for some programming langage (e.g. Python).
- Testing your script.
- Conclusion, visualization.

The two first steps obviously bring their share of potential errors but we will not deal with them for the moment. Grammatical and semantic errors have to do with the third and fourth steps.

Errors, debugging.

There are two different kinds of errors, you have the

- **grammatical errors** (also known as **syntactic errors**) that prevent your code from running
- and the **semantic errors**, i.e. when your code is running but not doing what you think it is.

The interpreter will automatically detect the first kind of errors. You need to detect on your own the second kind of errors.

We will list a few common syntactic errors.

In [2]:

```
# grammatical errors.
L=[]
# L[0] index is >len(L)-1
```

In [1]:

```
# semantic errors.
L=[1,3,2,4]
#Lsorted L.sort()#Wrong command
Lsorted=L.copy()#Good command
```

syntactic errors

A grammatical error arises when Python does not understand how a line of code should be interpreted. When such thing occurs, the interpreter will

- **stop** the code,
- point out which **lines** hasve led to the error (there might be multiple lines involved if you call functions),
- **name** the error
- and give an argument

Ideally, with the name of the mistake and the line of the mistake, we can debug the code.

You will find below a list of some common errors along with some advices to deal with them.

Neither the list of errors nor the list of advices is exhaustive.

Indentation Error

arises when a line of command is wrongly indented ("unexpected indent"), i.e. when you change the indentation level for a line which does not finish with a colon and which is not after a line ending with a colon.

It also arises when there is no indented line after a line ending with a colon ":" ("expected an indented block").

In [20]:

```
#Indentation Error
n=0
L=[]
while n<5: #pass
L.append(n)
    n=n+1
print(L)
```

```
  File "<ipython-input-20-e5170c46de4f>", line 5
    L.append(n)
    ^
IndentationError: expected an indented block
```

Actions to take.

- Check that all lines are vertically aligned.
- If all lines are visually aligned and you however get a IndentationError, it could be that you are mixing **invisible spaces** and **invisible tabs**. Whereas both look the same they are not the same and will lead to an error. To see these invisible characters, you should go to the setting page and tick the case "show invisible characters".
- Did you insert a (properly indented) line of command after a colon? If not, did you forget to do so or did you want the code to do nothing? If you want the code to do nothing after a colon then just erase the colon line as well.
- Be careful when you copy and past indented lines of code.

Index Error

when an index for a string/list is out of range.

Actions to take:

- Remember that the admissible indices for a list L goes from 0 to $\mathbf{len}(L)-1$, or from $-\mathbf{len}(L)$ to -1.
- Lists are mutable objects, check that the actions you take do not modify the length of the list. For instance you could use the **print** command and check if this is what you expect.
- If the index you are calling is a variable, are you sure you do not crucially modify the variable in your code?

In [21]:

```
#Example
L=[0,1,2]
L[3]
```

```
---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-21-d36c82155a1e> in <module>
      1 #Example
      2 L=[0,1,2]
----> 3 L[3]

IndexError: list index out of range
```

In [24]:

```
#Example
L=[0,1,2]
print(L[2])
L.remove(1)#after this operation the use of index 2 in L creates an error
print(L[2])
```

```
2

---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-24-844e80372733> in <module>
      3 print(L[2])
      4 L.remove(1)#after this operation the use of index 2 in L cre
ates an error
----> 5 print(L[2])

IndexError: list index out of range
```

Name Error

You use a name of variable which is not defined.

Action(s) to take.

- Localize the variable in the code using the line number given on the terminal.
- Localize the line (if any) where you defined the variable. Is it before/after the line where the error arises? Is it indented (if statement, while, for loops)?
- Remember that you need to write the exact same sequence of characters to call a variable. Did you make an orthograph or grammatical mistake when defining/calling the variable?
- If the line where you define or get the error is in a function, it might just be that your variable is local and your use is global or the other way around.
- If the name is used as a function, make sure you imported the relevant module before calling the function. Same thing with global constants such as π or γ.

In [25]:

```
#Example.
variable=0
while Variable<5: #even a mistake in the capitalisation can cause an error
    L.append(n)
    n=n+1
print(L)
```

```
---------------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-25-84e5bb89533e> in <module>
      1 #Example.
      2 variable=0
----> 3 while Variable<5: #even a mistake in the capitalisation can
 cause an error
      4     L.append(n)
      5     n=n+1

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

Overflow Error

arises when you reach the computational limit for floating numbers. It usually happens when you play around with very big integers and then convert them into floating numbers. The idea being that **integers can ask for a virtually infinite quantity of memory** (more precisely it will reach a **MemoryError** before giving up) whereas **floating numbers use only a limited amount of memory**.

In [5]:

```
# Example
n=(1.0*10**1000)/10**1000
print(n)
```

```
---------------------------------------------------------------------
-------
OverflowError                             Traceback (most recent cal
l last)
<ipython-input-5-72302de20f04> in <module>
      1 # Example
----> 2 n=(1.0*10**1000)/10**1000
      3 print(n)

OverflowError: int too large to convert to float
```

Runtime Error

arises when an error occurs but does not fall in any other type.

In [13]:

```
#Example
raise
```

```
---------------------------------------------------------------------
-------
RuntimeError                              Traceback (most recent cal
l last)
<ipython-input-13-308e86d4b02f> in <module>
      1 #Example
----> 2 raise

RuntimeError: No active exception to reraise
```

Syntax Error

arises when there is a problem of pure syntax.

For instance, it can arise when you use an invalid character in an identifier, it can also arises when you try to assign a value to a protected keyword or a number. Forgetting a colon after a **for**, **if**, **while**, **def**,... will also lead to a syntax error. The Python documentation refers to them as "errors" while any other Error is refered to as an Exception.

In [14]:

```
#Examples
#print(x
for x in range(0,9)
    print(x)
```

```
  File "<ipython-input-14-fdaf583d289c>", line 3
    for x in range(0,9)
                       ^
SyntaxError: invalid syntax
```

Action(s) to take.

- Use only latin letters without accent. Avoid accents and exotic letters.
- Strictly avoid the use of backslash, comas, colons,... That is all meaningful characters for Python.
- Usually when we write $1 = \ldots$ we actually mean $1 == \ldots$. Check that you did not mix the equality of assignement $=$ and the boolean equality $==$.
- Write the colon when you finish a **for**, **if**, **while**, **def**,.... Remark that with an enhanced text editor, when you put a colon at the end of a line, the line break automatically end up with an indented code. Should you forget to put a colon, you code will not be automatically indented. That's why, in general, it is hard to forget about the colon.

Type Error

arises when a wrong type is used. It typically happens when you call a built-in function that takes specific types as arguments. It also highlights a wrong number of arguments when calling a function. It could also arise when you use a string as an index for a list.

In [18]:

```
#Examples
L=['a','b','c']
#L['c']
#len(3)
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-18-7e59cae8b4f3> in <module>
      2 L=['a','b','c']
      3 #L['c']
----> 4 len(3)

TypeError: object of type 'int' has no len()
```

Action(s) to take.

- Look out for multiple affectations for one name of variable. Write longer names of variables to avoid potential conflicts.
- If you are using a built-in function, use the **help** function on the function to see which types and how many arguments you can give to the function.
- If you are using a built-in method for a specific type, use the **help** function of the type, to see why the method will not work (sometimes they have different names for different types).
- If applicable, change the type of your variables using a type function. You should test the outcome of these methods in the qpython interface before using it in a script.

Value Error

arises when you call a function with the right type but a forbidden value.

In [19]:

```
#Example
int('value')
#1/0
```

```
---------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-19-57fe6b21a365> in <module>
      1 #Example
----> 2 int('value')
      3 #1/0

ValueError: invalid literal for int() with base 10: 'value'
```

Action(s) to take.

- Call the **help** function to see if there is a forbidden value.
- When you divide by zero you directly get **ZeroDivisionError**. Look for small floating numbers that can be approximated by zero.
- In general, the problem comes from specific cases, e.g. when an integer is 0 or a list/ string of character is empty. You may have to consider **if** statements to specifically handle this case.