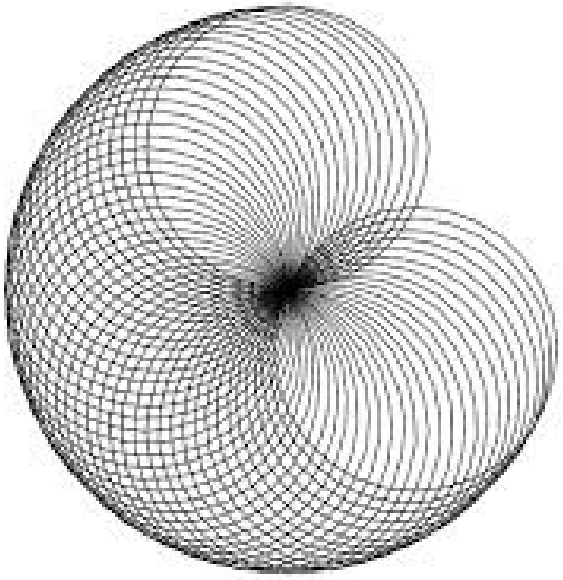
DA/AB

Fake Piscine

Week 1

"This is somewhere to be. This is all you have, but it's still something. Streets and sodium lights. The sky, the world. You're still alive."



INSTRUCTIONS

This is the start. Throughout this week you will learn - or recall - basic Python concepts. There are no rules: no norme, no moulinette, nothing. You will need to do what you are asked, and use any source you want to achieve this goal. You can deliver your project alone or in a group.

Create a general directory ("week1") and then a different directory for each exercise ("ex00", "ex01", etc..). Also add a .txt file including each person involved in the your team (or just your name if you're some kind of hermit).

ex00: print_sum.py

Write a program that prints the sum of two numbers inserted by the user. The program must be able to manage exceptions/input errors.

```
$> python3 print_sum.py 40 2
42
$>
```

ex01: print_secs

Write a program that lets the user insert a number of hours, minutes and seconds and prints the total number of seconds:

```
$> python3 print_secs.py 18 04 19
65059
$>
```

ex02: var

Create a script named "var.py" in which you will define a my_var function. In this function, you will declare 9 variables of different types and print them on the standard output. You will reproduce this output exactly:

```
$> python3 var.py
42 has a type <class 'int'>
42 has a type <class 'str'>
quarante-deux has a type <class 'str'>
42.0 has a type <class 'float'>
True has a type <class 'bool'>
[42] has a type <class 'list'>
{42: 42} has a type <class 'dict'>
(42,) has a type <class 'tuple'>
set() has a type <class 'set'>
$>
```

ex03: var_to_dict

Create a script named var_to_dict.py in which you will copy the following list of d couples as is in one of your functions:

```
d = [
('Hendrix' , '1942'),
('Allman' , '1946'),
('King' , '1925'),
('Clapton' , '1945'),
('Johnson' , '1911'),
('Berry' , '1926'),
('Vaughan' , '1954'),
('Cooder' , '1947'),
('Page' , '1944'),
('Richards' , '1943'),
('Hammett' , '1962'),
('Cobain' , '1967'),
('Garcia' , '1942'),
('Beck' , '1944'),
('Santana' , '1947'),
('Ramone' , '1948'),
('White' , '1975'),
('Frusciante', '1970'),
('Thompson' , '1949'),
('Burton' , '1939')
]
```

Your script must turn this variable into a dictionary. The year will be the key, the name of the musician the value. It must then display this dictionary on the standard output following a clear format:

```
1970 : Frusciante
1954 : Vaughan
1948 : Ramone
1944 : Page Beck
1911 : Johnson
...
```

ex04: capitals

Write a program that takes a state as an argument (ex: Oregon) and displays its capital city (ex: Salem) on the standard output. If the argument doesn't give any result, your script must display: Unknown state. If there is no argument - or too many - your script must no do anything and quit.

Dictionaries to copy unaltered:

```
states = {
"Oregon" : "OR",
"Alabama" : "AL",
"New Jersey": "NJ",
"Colorado" : "CO"
}
capital_cities = {
"OR": "Salem",
"AL": "Montgomery",
"NJ": "Trenton",
"CO": "Denver"
}
```

```
$> python3 capital_city.py Oregon
Salem
$> python3 capital_city.py Ile-De-France
Unknown state
$> python3 capital_city.py
$> python3 capital_city.py Oregon Alabama
$> python3 capital_city.py Oregon Alabama Ile-De-France
$>
```

ex05: value_search

Re-using the same dictionaries from the previous exercise, create a program that takes the capital city for argument and displays the matching state this time. The rest of your program's behaviors must remain the same as in the previous exercise.

```
$> python3 state.py Salem
Oregon
$> python3 state.py Paris
Unknown capital city
$> python3 state.py
$>
```

ex06: my_sort.py

Integrate the following dictionary in your code. Then write a program that displays the names of the musicians sorted by year in ascending order, then in alphabetic order for similar years. One per line, without showing the year.

```
d = {
'Hendrix' : '1942',
'Allman' : '1946',
'King' : '1925',
'Clapton' : '1945',
'Johnson' : '1911',
'Berry' : '1926',
'Vaughan' : '1954',
'Cooder' : '1947',
'Page' : '1944',
'Richards' : '1943',
'Hammett' : '1962',
'Cobain' : '1967',
'Garcia' : '1942',
'Beck' : '1944',
'Santana' : '1947',
'Ramone' : '1948',
'White' : '1975',
'Frusciante': '1970',
'Thompson' : '1949',
'Burton' : '1939',
}
```

ex07: my_first_class

Create a file called "intern.py". Then create a Intern class containing the following functionalities:
A builder taking a character chain as a parameter, assigning its value to a Name attribute. "My name? I’m nobody, an intern, I have no name." will be implemented as default value.
A method __str__() that will return Name attribute of the instance.
A Coffee class with a simple __str__() method that will return the character chain "This is the worst coffee you ever tasted.".
A work() method that will raise only one exception (use the basic (Exception) type)
with the text "I’m just an intern, I can’t do that...".
A make_coffee() method that will return an instance of the Coffee class that you will have implemented in Intern class.
In your tests, you will have to instantiate twice the Intern class. Once without a name, the second time with the name "Mark".
Display the name of each instance. Ask mark to make you a coffee and display the result. Ask the other intern to work. You will have to manage the exception in your test.

ex08: biv', famm' capi

Create a file called beverages.py. Inside create a HotBeverage class with the following functionalities:
- A price attribute with the value of 0.30
- A name attribute with the "hot beverage" value
- A description() method returning an instance description. The description value will be "Just some hot water in a cup".
-A __str__() method returning an instance description in this form

```
name : <name attribute>
price : <price attribute limited to two decimal points>
description : <instance's description>
```

for instance, a HotBeverage instance display would look like this:

```
name : hot beverage
price : 0.30
description : Just some hot water in a cup.
```

Then create the following derived classes from HotBeverage:
- Coffee:
    name: "coffee"
    price: 0.40
    description: "A coffee, to stay awake."
- Tea:
    name: "tea"

```
    price: 0.30
    description: "Just some hot water in a cup."
- Chocolate:
    name: "chocolate"
    price: 0.50
    description: "Chocolate, sweet chocolate..."
- Cappuccino:
    name: "cappuccino"
    price: 0.45
    description: "Un po' di Italia nella sua tazza!"
```

In your tests, instantiate each class among: HotBeverage, Coffee, Tea, Chocolate and Cappuccino and display them.

ex09: where_is_my_coffee

You will upload two files(machine.py and beverages.py).
Create the CoffeeMachine class containing:
• A builder.
• An EmptyCup class inheriting HotBeverage, with the name "empty cup", a 0.90 price and the description "An empty cup?! Gimme my money back!".
Copy the beverages.py file frome the previous exercise in this exercise folder to use the classes it contains.
• A BrokenMachineException class inheriting the Exception with the text "This coffee machine has to be repaired.". This text must be defined in the exception's builder.
• A repair() method that fixes the machine so it can serve hot drinks again.
• A serve() method that will have the following specifications:
Parameters : A unique parameter (other than self) that will be a class derived from HotBeverage.
Return: Alternatively (randomly), the method returns an instance of the class set in parameter and alternatively, an EmptyCup instance. Obsolescence: The machine is cheap and breaks down after serving 10 drinks. When out of order: the call for the serve() method must raise a CoffeeMachine.BrokenMachitype exception until the repair() method is called.

Fixing: After calling the repair() method, the serve() method can work again without raising the exception before it breaks down again after serving 10 drinks. In your tests, instantiate the CoffeeMachine class. Ask for various drinks from the beverages.py file and display the drink you're served until the machine breaks down (you will then manage the raised exception). Fix the machine and start again until the machine breaks down again (manage the exception again).