Piscine 101

Python 00

Summary: This document is the subject for the PYTHON module 00 of the Piscine 101 @ 42Tokyo.

Contents

I Instructions 2

II Foreword 4

III Exercise 00 : Hello from python! 5

IV Exercise 01 : my first variables 7

V Exercise 02 : FizzBuzz 8

VI Exercise 03 : Numbers 10

VII Exercise 04 : My first dictionary 11

VIII Exercise 05 : Key search 13

IX Exercise 06 : Search by value 14

X Exercise 07 : Search by key or value 15

XI Exercise 08 : Create your own command 17

Chapter I

Instructions

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- On top of that, your exercises will be checked and graded by a program called Moulinette.
- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible.
- Exercises in Shell must be executable with /bin/sh.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ...`.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- If no other explicit information is displayed, you must assume the following versions of languages : `Python - python3.8.2`.
- No code in the global scope(only import). We want functions!
- you're free to define as many function as you like and name them as you like also.

- Each turned-in file must end with a function call in a condition identical to:

```
if __name__ == '__main__':
    your_function( whatever, parameter, is, required )
```

- You can set an error management in this condition.
- No import will be authorized, except the ones explicitly mentioned in the 'Autorized functions' in each exercise's description.
- You won't have to manage the exceptions raised by the `open` function.

Chapter II

Foreword

`https://xkcd.com/353/`

Chapter III

Exercise 00 : Hello from python!

	Exercise 00
Only the best know how to print Hello	
Turn-in directory : *ex*00/	
Files to turn in : `hello.py`	
Allowed functions : `n/a`	

- Create a script named `hello.py` in which you will define a `print_hello` function. In this function, you will print 'Hello!' on the standard output. You will reproduce this output exactly:
- Check the example and follow the same format for the output of the command.

  Example:

```
?>python3 hello.py
Hello!
?>
```

`https://bit.ly/3tkznIU`

`https://bit.ly/2MGe2Jb`

https://bit.ly/3tsTAMW

Chapter IV

Exercise 01 : my first variables

	Exercise 01
my first variables	
Turn-in directory : *ex*01/	
Files to turn in : `var.py`	
Allowed functions : `n/a`	

- 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'>
forty-two 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'>
?>
```

- Of course, explicitly writing your variable types in the prints of you code is strictly prohibited. Don't forget to call your function at the end of your script as required by the instructions:

```
if __name__ == '__main__':
    my_var()
```

`https://bit.ly/3cykcpu`

Chapter V

Exercise 02 : FizzBuzz

	Exercise 02
FizzBuzz	
Turn-in directory : *ex*02/	
Files to turn in : `fizzbuzz.py`	
Allowed functions : `sys`	

- Write a python3 script that prints the numbers from 1 to the number given from command line argument, each separated by a newline.
- If the number is a multiple of 3, it prints 'fizz' instead.
- If the number is a multiple of 5, it prints 'buzz' instead.
- If the number is both a multiple of 3 and a multiple of 5, it prints 'fizzbuzz' instead.
- If there is no command line argument or more than one command line argument, do nothing.
- If there is one command line argument, the value will always be between 1 and 100.

  Example:

```
%> python3 fizzbuzz.py 16 | cat -e
1$
2$
fizz$
4$
buzz$
fizz$
7$
8$
fizz$
buzz$
11$
fizz$
13$
14$
fizzbuzz$
16$
%>
%> python3 fizzbuzz.py | cat -e
```

```
%>
%> python3 fizzbuzz.py 16 100 | cat -e
%>
```

https://bit.ly/39JsskR

Chapter VI

Exercise 03 : Numbers

	Exercise 03
Numbers	
Turn-in directory : *ex*03/	
Files to turn in : `numbers.py`	
Allowed functions : `n/a`	

- The resources.tar.gz tarball in the appendix of this subject contains a ex03/ subfolder that holds a numbers.txt file containing the numbers 1 to 100 separated by a coma
- Create a Python script named `numbers.py` which role is to open a `numbers.txt` file, read the numbers it contains and display them on the standard output, one per line, without any coma.

`https://bit.ly/3cEA8qm`

Chapter VII

Exercise 04 : My first dictionary

	Exercise 04
My first dictionary	
Turn-in directory : *ex*04/	
Files to turn in : `var_to_dict.py`	
Allowed functions : `n/a`	

- 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 = [
  ('Allman' , '1946'),
  ('King' , '1925'),
  ('Clapton' , '1945'),
  ('Johnson' , '1911'),
  ('Berry' , '1926'),
  ('Vaughan' , '1954'),
  ('Cooder' , '1947'),
  ('Richards' , '1943'),
  ('Hammett' , '1962'),
  ('Cobain' , '1967'),
  ('Garcia' , '1942'),
  ('Beck' , '1944'),
  ('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:

```
1946: Allman
1925: King
1945: Clapton
1911: Johnson
1926: Berry
1954: Vaughan
1947: Cooder
1943: Richards
1962: Hammett
...
```

https://bit.ly/2Muc12V

Chapter VIII

Exercise 05 : Key search

	Exercise 05
Key search	
Turn-in directory : *ex*05/	
Files to turn in : `capital_city.py`	
Allowed functions : `sys`	

- Here are dictionaries you have to copy unaltered in one of your script's functions:

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

- 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.

```
$> 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
$>
```

https://bit.ly/2Muc12V

Chapter IX

Exercise 06 : Search by value

	Exercise 06
Search by value	
Turn-in directory : *ex*06/	
Files to turn in : `state.py`	
Allowed functions : `sys`	

- Here are dictionaries you have to copy unaltered in one of your script's functions:

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

- Create a program that takes the capital city for argument and displays the matching state this time. If the argument doesn't give any result, your script must display: `Unknown capital city`. 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
$>
```

`https://bit.ly/2Muc12V`

Chapter X

Exercise 07 : Search by key or value

	Exercise 07
Search by key or value	
Turn-in directory : *ex*07/	
Files to turn in : `all_in.py`	
Allowed functions : `sys`	

- Here are dictionaries you have to copy unaltered in one of your script's functions:

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

- The program must take for argument a string containing as many expressions as we search for, separated by a coma.
- For each expression in this string, the program must detect if it's a capital, a state or none of them.
- The program must not be case-sensitive. It must not take multiple spaces in consideration either.
- If there is no parameter or too many parameters, the program doesn't display anything.
- Check the example and follow the same format for the output of the command.

  Example:

```
$> python3 all_in.py "New jersey, Tren ton, NewJersey, Trenton, toto, , sAlem"
Trenton is the capital of New Jersey
```

```
Tren ton is neither a capital city nor a state
NewJersey is neither a capital city nor a state
Trenton is the capital of New Jersey
toto is neither a capital city nor a state
Salem is the capital of Oregon
$>
```

https://bit.ly/2Muc12V

Chapter XI

Exercise 08 : Create your own command

	Exercise 08
Create your own command	
Turn-in directory : *ex*08/	
Files to turn in : `file_viewer.py`	
Allowed functions : `sys, argparse`	

- Write a python script that prints the file's content from a specified line to another specified line.
- Check the example and follow the same format for the output of the command.

  Example:

```
%> python3 file_viewer.py --help | cat -e
usage: file_viewer.py [-h] filename start end$
$
Display specific part of the file.$
$
positional arguments:$
  filename   filename$
  start      starting line$
  end        ending line$
$
optional arguments:$
  -h, --help show this help message and exit$
%>
%> python3 file_viewer.py /etc/passwd 11 13 | cat -e
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false$
root:*:0:0:System Administrator:/var/root:/bin/sh$
daemon:*:1:1:System Services:/var/root:/usr/bin/false$
%>
%> python3 file_viewer.py /etc/passwd 11 13 | cat -e
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false$
root:*:0:0:System Administrator:/var/root:/bin/sh$
daemon:*:1:1:System Services:/var/root:/usr/bin/false$
%>
%> python3 file_viewer.py /etc/passwd 11 a | cat -e
usage: file_viewer.py [-h] filename start end
file_viewer.py: error: argument end: invalid int value: 'a'
%>
%> python3 file_viewer.py /etc/passwd a 13 | cat -e
```

```
usage: file_viewer.py [-h] filename start end
file_viewer.py: error: argument end: invalid int value: 'a'
%>
%> python3 file_viewer.py /etc/passwd | cat -e
usage: file_viewer.py [-h] filename start end
file_viewer.py: error: the following arguments are required: filename, start, end
%>
%> python3 file_viewer.py nosuchfile 11 13 | cat -e
file_viewer.py: nosuchfile: No such file$
%>
%> python3 file_viewer.py /etc/passwd 1000 1001
%>
%> python3 file_viewer.py /etc/passwd 10 1
%>
%> python3 file_viewer.py | cat -e
usage: file_viewer.py [-h] filename start end
file_viewer.py: error: the following arguments are required: filename, start, end
%>
```

https://bit.ly/3tpqrC3