0x00. Python - Hello, World

`Python`

- By: Guillaume
- Weight: 1
- Project over - took place from May 30, 2022 6:00 AM to May 31, 2022 6:00 AM
- An auto review will be launched at the deadline

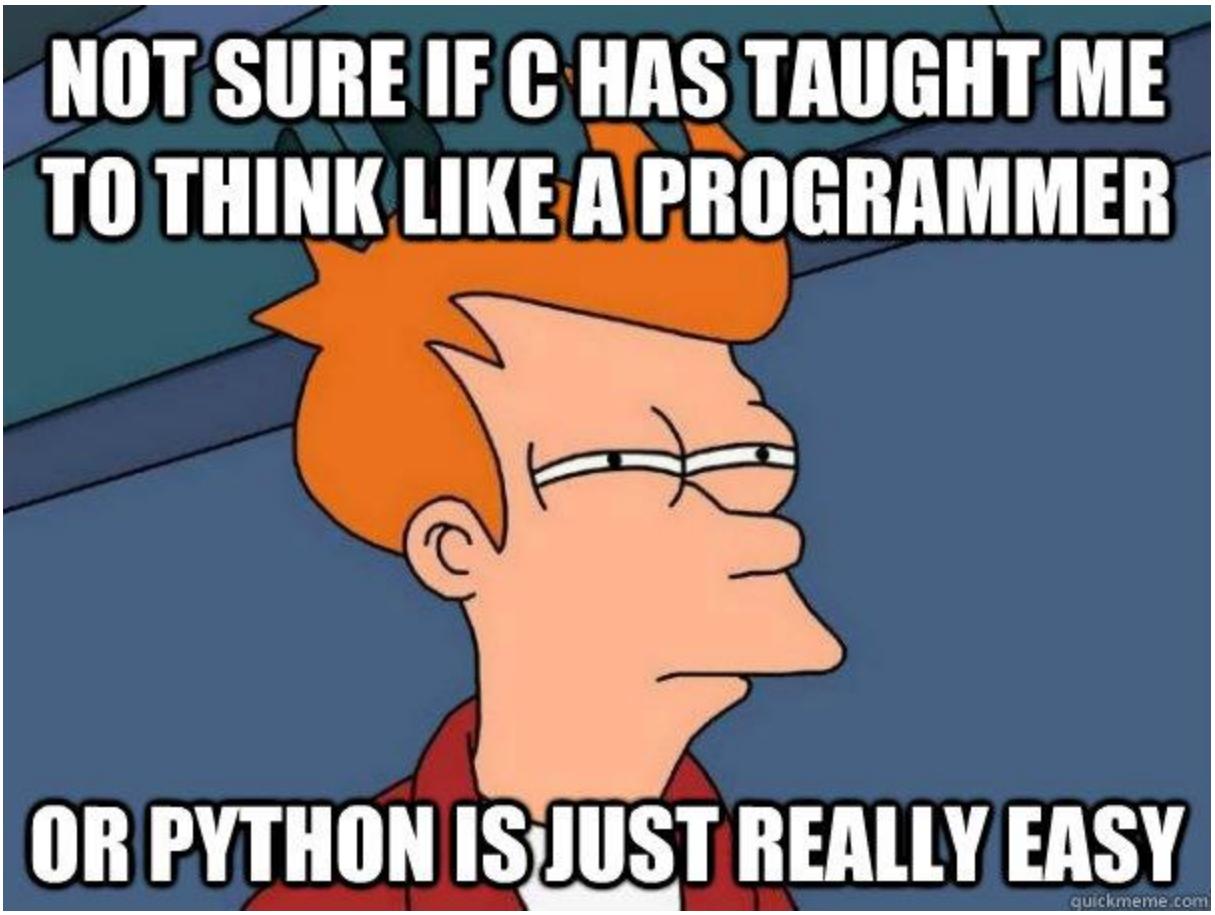
In a nutshell…

- **Auto QA review:** 57.85/89 mandatory & 14.3/27 optional
- **Altogether: 99.42%**
  - Mandatory: 65.0%
  - Optional: 52.96%
  - Calculation: 65.0% + (65.0% * 52.96%) == **99.42%**

Concepts

For this project, we expect you to look at this concept:

- Python programming



Author’s disclaimer

```
Welcome to the Python world!


The first projects are more "C-oriented" - no tricks, no funky syntax - simple!

If you've already played with Python, don't worry, fun things will come.

You'll soon find that with Python (and the majority of higher level languages), there
are ten different ways to do the same thing. Some tasks will expect only one implemen
tation, while other tasks will have multiple possible implementations.

Like C, Python also has a linter / style guide like Betty, called PEP8, also now know
n as PyCode.


Enjoy!


- Guillaume
```

Resources

Read or watch:

- The Python tutorial (*only the first three chapters below*)
- Whetting Your Appetite
- Using the Python Interpreter
- An Informal Introduction to Python (*Read up until "3.1.2. Strings" included*)
- How To Use String Formatters in Python 3
- Learn to Program
- Pycodestyle – Style Guide for Python Code

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone, **without the help of Google**:

General

- Why Python programming is awesome
- Who created Python
- Who is Guido van Rossum
- Where does the name 'Python' come from
- What is the Zen of Python
- How to use the Python interpreter
- How to print text and variables using `print`
- How to use strings
- What are indexing and slicing in Python
- What is the official Python coding style and how to check your code with `pycodestyle`

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

Python Scripts

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be interpreted/compiled on Ubuntu 20.04 LTS using python3 (version 3.8.5)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/usr/bin/python3`
- A `README.md` file at the root of the repo, containing a description of the repository
- A `README.md` file, at the root of the folder of *this* project, is mandatory
- Your code should use the pycodestyle (version `2.8.*`)
- All your files must be executable
- The length of your files will be tested using `wc`

Shell Scripts

- Allowed editors: `vi`, `vim`, `emacs`
- All your scripts will be tested on Ubuntu 20.04 LTS
- All your scripts should be exactly two lines long (`wc -l file` should print 2)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/bin/bash`
- All your files must be executable

C Scripts

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using gcc, using the options -Wall -Werror -Wextra -pedantic -std=gnu89
- All your files should end with a new line
- Your code should use the `Betty` style. It will be checked using betty-style.pl and betty-doc.pl
- You are not allowed to use global variables
- No more than 5 functions per file
- In the following examples, the `main.c` files are shown as examples. You can use them to test your functions, but you don't have to push them to your repo (if you do we won't take them into account). We will use our own `main.c` files at compilation. Our `main.c` files might be different from the one shown in the examples
- The prototypes of all your functions should be included in your header file called `lists.h`
- Don't forget to push your header file
- All your header files should be include guarded

More Info

Zen

```
The Zen of Python, by Tim Peters


Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Pycodestyle

`Pycodestyle` is now the new standard of Python style code

Quiz questions

Great! You've completed the quiz successfully! Keep going! (Show quiz)

Tasks

0. Run Python file

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a Shell script that runs a Python script.

The Python file name will be saved in the environment variable `$PYFILE`

```
guillaume@ubuntu:~/py/0x00$ cat main.py
#!/usr/bin/python3
print("Best School")

guillaume@ubuntu:~/py/0x00$ export PYFILE=main.py
guillaume@ubuntu:~/py/0x00$ ./0-run
Best School
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `0-run`

Done! Help Check your code Get a sandbox QA Review

1. Run inline

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a Shell script that runs Python code.

The Python code will be saved in the environment variable `$PYCODE`

```
guillaume@ubuntu:~/py/0x00$ export PYCODE='print(f"Best School: {88+10}")'
guillaume@ubuntu:~/py/0x00$ ./1-run_inline
Best School: 98
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `1-run_inline`

Done! Help Check your code Get a sandbox QA Review

2. Hello, print

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a Python script that prints exactly `"Programming is like building a multilingual puzzle`, followed by a new line.

- Use the function `print`

```
guillaume@ubuntu:~/py/0x00$ ./2-print.py
"Programming is like building a multilingual puzzle
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `2-print.py`

Done! Help Check your code Get a sandbox QA Review

3. Print integer

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Complete this source code in order to print the integer stored in the variable `number`, followed by `Battery street`, followed by a new line.

- You can find the source code here
- The output of the script should be:
  - the number, followed by `Battery street`,
  - followed by a new line
- You are not allowed to cast the variable `number` into a string
- Your code must be 3 lines long
- You have to use f-strings tips

```
guillaume@ubuntu:~/py/0x00$ ./3-print_number.py
98 Battery street
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `3-print_number.py`

Done! Help Check your code Get a sandbox QA Review

4. Print float

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Complete the source code in order to print the float stored in the variable `number` with a precision of 2 digits.

- You can find the source code here
- The output of the program should be:
  - `Float:`, followed by the float with only 2 digits
  - followed by a new line
- You are not allowed to cast `number` to string
- You have to use f-strings

```
guillaume@ubuntu:~/py/0x00$ ./4-print_float.py
Float: 3.14
```

```
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `4-print_float.py`

Done! Help Check your code Get a sandbox QA Review

5. Print string

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Complete this source code in order to print 3 times a string stored in the variable `str`, followed by its first 9 characters.

- You can find the source code here
- The output of the program should be:
  - 3 times the value of `str`
  - followed by a new line
  - followed by the 9 first characters of `str`
  - followed by a new line
- You are not allowed to use any loops or conditional statement
- Your program should be maximum 5 lines long

```
guillaume@ubuntu:~/py/0x00$ ./5-print_string.py
Holberton SchoolHolberton SchoolHolberton School
Holberton
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `5-print_string.py`

Done! Help Check your code Get a sandbox QA Review

6. Play with strings

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Complete this source code to print `Welcome to Holberton School!`

- You can find the source code here
- You are not allowed to use any loops or conditional statements.
- You have to use the variables `str1` and `str2` in your new line of code
- Your program should be exactly 5 lines long

```
guillaume@ubuntu:~/py/0x00$ ./6-concat.py
Welcome to Holberton School!
guillaume@ubuntu:~/py/0x00$ wc -l 6-concat.py
5 6-concat.py
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `6-concat.py`

Done! Help Check your code Get a sandbox QA Review

7. Copy - Cut - Paste

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Complete this source code

- You can find the source code here
- You are not allowed to use any loops or conditional statements
- Your program should be exactly 8 lines long
- `word_first_3` should contain the first 3 letters of the variable `word`
- `word_last_2` should contain the last 2 letters of the variable `word`
- `middle_word` should contain the value of the variable `word` without the first and last letters

```
guillaume@ubuntu:~/py/0x00$ ./7-edges.py
First 3 letters: Hol
Last 2 letters: on
Middle word: olberto
guillaume@ubuntu:~/py/0x00$ wc -l 7-edges.py
8 7-edges.py
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `7-edges.py`

Done! Help Check your code Get a sandbox QA Review

8. Create a new sentence

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Complete this source code to print `object-oriented programming with Python`, followed by a new line.

- You can find the source code here
- You are not allowed to use any loops or conditional statements
- Your program should be exactly 5 lines long
- You are not allowed to create new variables
- You are not allowed to use string literals

```
guillaume@ubuntu:~/py/0x00$ ./8-concat_edges.py
object-oriented programming with Python
guillaume@ubuntu:~/py/0x00$ wc -l 8-concat_edges.py
5 8-concat_edges.py
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `8-concat_edges.py`

Done! Help Check your code Get a sandbox QA Review

9. Easter Egg

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a Python script that prints “The Zen of Python”, by TimPeters, followed by a new line.

- Your script should be maximum 98 characters long (please check with `wc -m 9-easter_egg.py`)

```
guillaume@ubuntu:~/py/0x00$ ./9-easter_egg.py
The Zen of Python, by Tim Peters
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `9-easter_egg.py`

Done! Help Check your code Get a sandbox QA Review

10. Linked list cycle

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Technical interview preparation:

- You are not allowed to google anything
- Whiteboard first

- This task and all future technical interview prep tasks will include checks for the efficiency of your solution, i.e. is your solution's runtime fast enough, does your solution require extra memory usage / mallocs, etc.

Write a function in C that checks if a singly linked list has a cycle in it.

- Prototype: `int check_cycle(listint_t *list);`
- Return: `0` if there is no cycle, `1` if there is a cycle

Requirements:

- Only these functions are allowed: `write`, `printf`, `putchar`, `puts`, `malloc`, `free`

```
carrie@ubuntu:~/0x00$ cat lists.h
#ifndef LISTS_H
#define LISTS_H

#include <stdlib.h>

/**
 * struct listint_s - singly linked list
 * @n: integer
 * @next: points to the next node
 *
 * Description: singly linked list node structure
 *
 */
typedef struct listint_s
{
    int n;
    struct listint_s *next;
} listint_t;

size_t print_listint(const listint_t *h);
listint_t *add_nodeint(listint_t **head, const int n);
void free_listint(listint_t *head);
int check_cycle(listint_t *list);
```

```
#endif /* LISTS_H */
carrie@ubuntu:~/0x00$ cat 10-linked_lists.c
#include <stdio.h>
#include <stdlib.h>
#include "lists.h"

/**
 * print_listint - prints all elements of a listint_t list
 * @h: pointer to head of list
 * Return: number of nodes
 */
size_t print_listint(const listint_t *h)
{
    const listint_t *current;
    unsigned int n; /* number of nodes */

    current = h;
    n = 0;
    while (current != NULL)
    {
        printf("%i\n", current->n);
        current = current->next;
        n++;
    }

    return (n);
}

/**
 * add_nodeint - adds a new node at the beginning of a listint_t list
 * @head: pointer to a pointer of the start of the list
 * @n: integer to be included in node
```

```
 * Return: address of the new element or NULL if it fails
 */
listint_t *add_nodeint(listint_t **head, const int n)
{
    listint_t *new;

    new = malloc(sizeof(listint_t));
    if (new == NULL)
        return (NULL);

    new->n = n;
    new->next = *head;
    *head = new;

    return (new);
}

/**
 * free_listint - frees a listint_t list
 * @head: pointer to list to be freed
 * Return: void
 */
void free_listint(listint_t *head)
{
    listint_t *current;

    while (head != NULL)
    {
        current = head;
        head = head->next;
        free(current);
    }
}
```

```
carrie@ubuntu:~/0x00$ cat 10-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    listint_t *current;
    listint_t *temp;
    int i;

    head = NULL;
    add_nodeint(&head, 0);
    add_nodeint(&head, 1);
    add_nodeint(&head, 2);
    add_nodeint(&head, 3);
    add_nodeint(&head, 4);
    add_nodeint(&head, 98);
    add_nodeint(&head, 402);
    add_nodeint(&head, 1024);
    print_listint(head);

    if (check_cycle(head) == 0)
        printf("Linked list has no cycle\n");
    else if (check_cycle(head) == 1)
        printf("Linked list has a cycle\n");
```

```
    current = head;
    for (i = 0; i < 4; i++)
        current = current->next;
    temp = current->next;
    current->next = head;

    if (check_cycle(head) == 0)
        printf("Linked list has no cycle\n");
    else if (check_cycle(head) == 1)
        printf("Linked list has a cycle\n");

    current = head;
    for (i = 0; i < 4; i++)
        current = current->next;
    current->next = temp;

    free_listint(head);

    return (0);
}
carrie@ubuntu:~/0x00$ gcc -Wall -Werror -Wextra -pedantic -std=gnu89 10-main.c 10-che
ck_cycle.c 10-linked_lists.c -o cycle
carrie@ubuntu:~/0x00$$ ./cycle
1024
402
98
4
3
2
1
0
Linked list has no cycle
Linked list has a cycle
```

```
carrie@ubuntu:~/0x00$
```

Solving a problem is already a big win! but finding the best and optimal way to solve it, it's way better! Think about the most optimal / fastest way to do it.

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `10-check_cycle.c, lists.h`

Done! Help Check your code Get a sandbox QA Review

11. Hello, write

#advanced

Score: 65.0% (*Checks completed: 100.0%*)

Write a Python script that prints exactly `and that piece of art is useful - Dora Korpar, 2015-10-19`, followed by a new line.

- Use the function `write` from the `sys` module
- You are not allowed to use `print`
- Your script should print to `stderr`
- Your script should exit with the status code `1`

```
guillaume@ubuntu:~/py/0x00$ ./100-write.py
and that piece of art is useful - Dora Korpar, 2015-10-19
guillaume@ubuntu:~/py/0x00$ echo $?
1
guillaume@ubuntu:~/py/0x00$ ./100-write.py 2> q
guillaume@ubuntu:~/py/0x00$ cat q
and that piece of art is useful - Dora Korpar, 2015-10-19
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `100-write.py`

Done! Help Check your code Get a sandbox QA Review

12. Compile

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a script that compiles a Python script file.

The Python file name will be stored in the environment variable `$PYFILE`

The output filename has to be `$PYFILEc` (ex: `export PYFILE=my_main.py` => output filename: `my_main.pyc`)

```
guillaume@ubuntu:~/py/0x00$ cat main.py
#!/usr/bin/python3
print("Best School")

guillaume@ubuntu:~/py/0x00$ export PYFILE=main.py
guillaume@ubuntu:~/py/0x00$ ./101-compile
Compiling main.py ...
guillaume@ubuntu:~/py/0x00$ ls
101-compile  main.py  main.pyc
guillaume@ubuntu:~/py/0x00$ cat main.pyc | zgrep -c "Best School"
1
guillaume@ubuntu:~/py/0x00$ od -t x1 main.pyc # SYSTEM DEPENDANT => CAN BE DIFFERENT
0000000 ee 0c 0d 0a 91 26 3e 58 31 00 00 00 e3 00 00 00
0000020 00 00 00 00 00 00 00 00 02 00 00 00 40 00 00 00
0000040 00 73 0e 00 00 00 65 00 00 64 00 00 83 01 00 01
0000060 64 01 00 53 29 02 7a 10 48 6f 6c 62 65 72 74 6f
0000100 6e 20 53 63 68 6f 6f 6c 4e 29 01 da 05 70 72 69
0000120 6e 74 a9 00 72 02 00 00 00 72 02 00 00 00 fa 07
0000140 6d 61 69 6e 2e 70 79 da 08 3c 6d 6f 64 75 6c 65
0000160 3e 02 00 00 00 73 00 00 00 00
0000172
guillaume@ubuntu:~/py/0x00$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `101-compile`

Done! Help Check your code Ask for a new correction Get a sandbox QA Review

13. ByteCode -> Python #1

#advanced

Score: 65.0% (*Checks completed: 100.0%*)

Write the Python function `def magic_calculation(a, b):` that does exactly the same as the following Python bytecode:

```
  3           0 LOAD_CONST               1 (98)
              3 LOAD_FAST                0 (a)
              6 LOAD_FAST                1 (b)
              9 BINARY_POWER
             10 BINARY_ADD
             11 RETURN_VALUE
```

- Tip: Python bytecode

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x00-python-hello_world`
- File: `102-magic_calculation.py`

Done! Help Check your code Get a sandbox QA Review