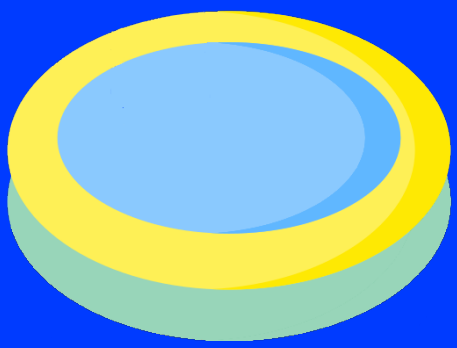
{EPITECH}

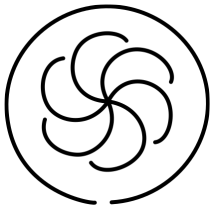
WADING POOL_

< 09 - ADVANCED FUNCTIONS + ERRORS + FILES />



WADING POOL

Code wars
In addition to the tasks below, you must go as far as possible in this code wars collection. Try to solve the first one until the last one without skipping challenges!
Work on it as soon as you have a bit of time, or whenever you need a break in you day!



Your functions should handle errors.
For instances: no argument, not enough arguments, unknown arguments, bad type, ...

Have a closer look at python **exceptions**.

Deeper within functions

Task 1.1

Create a function `my_sum` that accepts a variable number of arguments and print their sum.

For instance:

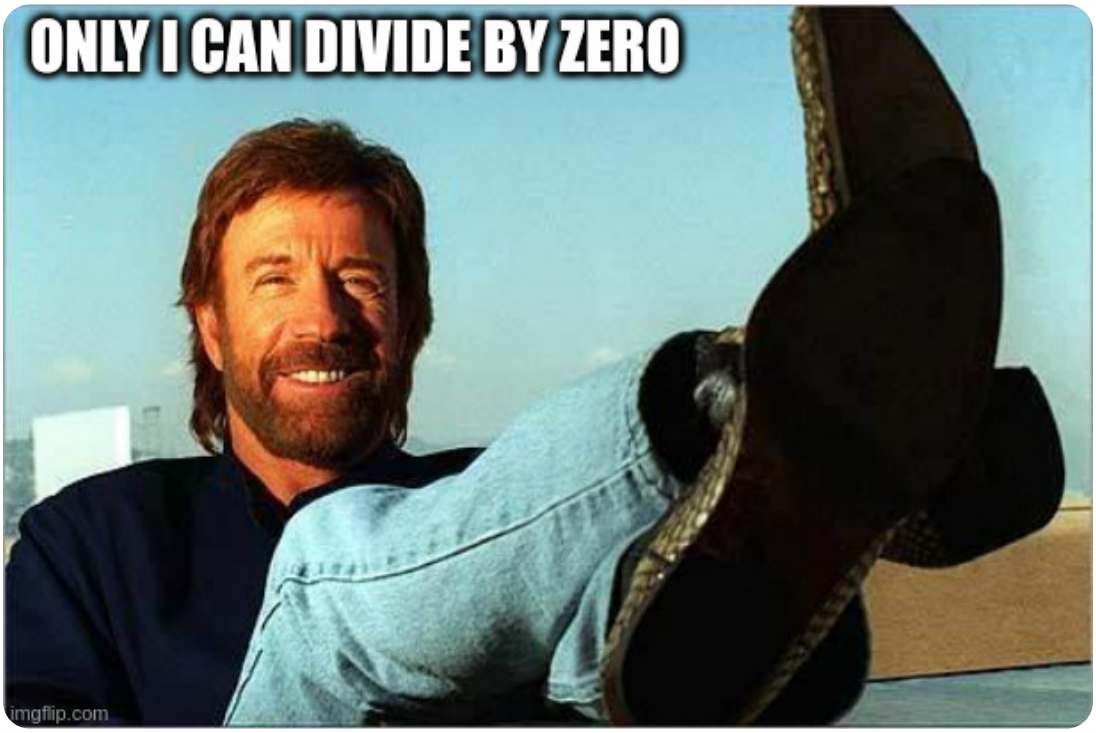
- ✓ `my_sum(1)` returns `1` ;
- ✓ `my_sum(1,2,3)` returns `6` ;
- ✓ `my_sum(-20, -10, 5, 5, 10, 10)` returns `0` ;
- ✓ `my_sum(1, "toto")` throws an *Exception ValueError* but no *Traceback*.

Task 1.2

Write a function `my_division` that takes two int as parameters, and returns **both** the quotient and the remainder of the euclidean division of the first by the second parameter. It should looks like:

```
Terminal
$> my_division(42 , 4)
10
2
```

You're function must handle errors.



Task 1.3

Create the function `my_count` that complies with the following conditions:

- ✓ it takes at least one parameter, named `stop`, which is an integer ;
- ✓ it can accept a second argument, named `start`, which is also an integer ;
- ✓ if second argument is missing in the function call, assign the **default** value `0` to `start` ;
- ✓ it prints all integers from `start` to `stop`.

Task 1.4

Upgrade your previous function with a second default argument, named `step`, which is an integer. If not provided, its default value is `1`. The function prints all integers from `start` (included) to `stop` (excluded).

Your function handles as many cases as possible.
Your function should handles errors.

- ✓ What if step = 0?
- ✓ What if stop < start?
- ✓ What if stop = ”toto”?

Terminal

```
$> my count(100, -100, 42)
-100
-58
-16
26
68
```

Terminal

```
$> my_count(-100, 100, -42)
100
58
16
-26
-68
```

Task 1.5

Create a function `new_division` that:

- ✓ takes two arguments, `num` and `den`, which are numbers ;
- ✓ accepts an optional third argument, `acc`, which is an integer (default = 1) ;
- ✓ prints the result of the division of `num` by `den` with an accuracy of `acc` digits after 0.

For instances, `new_division(8.4, 13)=0.6` and `new_division(8.4, 13, 6)=0.646154`.

Task 1.6

Create a `ship` function that:

- ✓ takes a variable number of arguments, corresponding to the fullname of somebody ;
- ✓ takes a variable number of **keyword arguments**, corresponding to the full address ;
- ✓ prints the full shipping label in order to mail something to this person.

For instance:

```
Terminal
$> ship("Batman", street="Mountain Drive", city="Gotham")
Batman
street: Mountain Drive
city: Gotham
```

```
Terminal
$> ship("Superman", "The man of steel", apartment="3D", num= 344, street="Clinton Street",
city="Metropolis")
Superman The man of steel
apartment: 3D
num: 344
street: Clinton Street
city: Metropolis
```

CHALLENGE

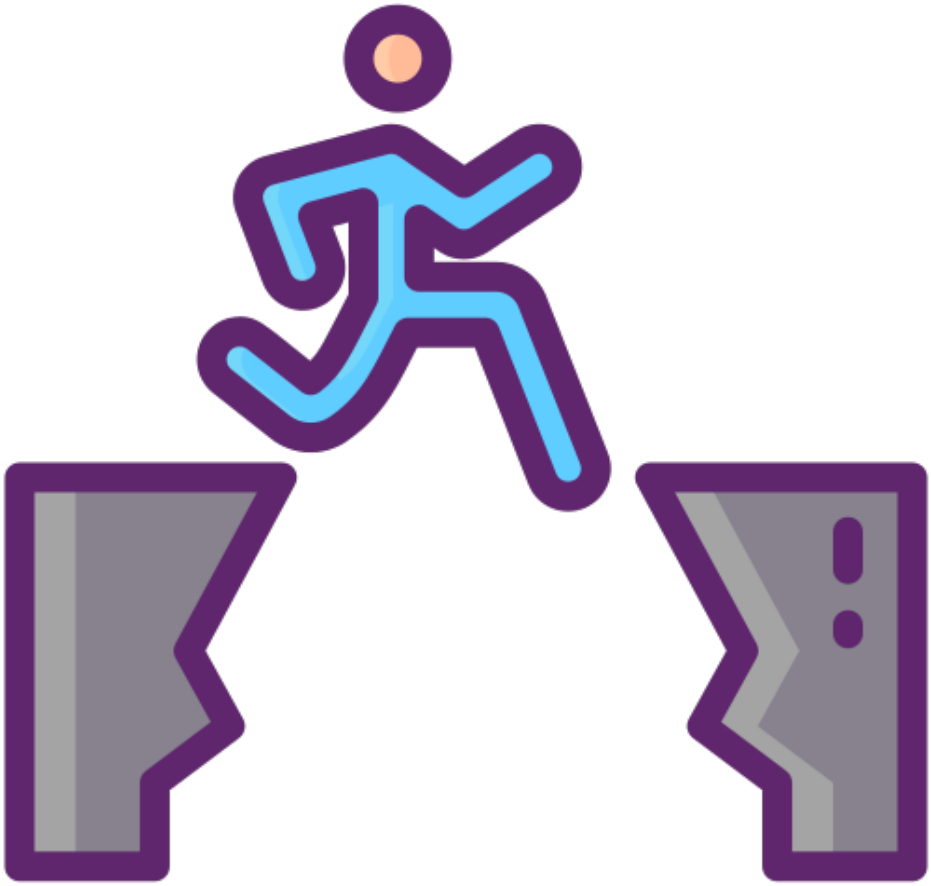
Create a program that takes an integer `N` as argument and count how many letters would be used if all the numbers from 1 to `N` (included) were written out in words.

Spaces and hyphens should NOT be counted.
The use of ”and” when writing out numbers is in compliance with British usage.
For instance:

- ✓ 42 (forty-two) contains 8 letters ;
- ✓ 115 (one hundred and fifteen) contains 20 letters.

To guide you in your quest, here are some inputs and corresponding outputs:

- ✓ N = 5 returns 19 (one, two, three, four, five => 3 + 3 + 5 + 4 + 4 = 19) ;
- ✓ N = 42 returns 319 ;
- ✓ N = 1000 returns 21124.



Files manipulation

For the following exercises, you'll need two files:

- ✓ The zen of Python ;
- ✓ The table of n, prime(n) for n = 1..10000 from OEIS.org.

Save them in your current working directory with the names `zen.txt` and `primes.txt`.

Your functions should handle errors (non-existing file, unreadable file, empty file, ...etc).
Before reading a file, you need to open it. After reading a file, you should close it.

Have a closer look at python **exceptions**.

Task 2.1

Write a `read_file` function that reads and prints **entirely** the file `primes.txt`.

Task 2.2

Write a `read_line` function that reads and prints **line by line** the file `zen.txt`.

Task 2.3

Write a `read_lines` function that:

- ✓ take a variable number of integers as parameter ;
- ✓ then reads and prints the corresponding line(s) from `primes.txt`.

For instance, `read_lines(666)` returns `666 4973` and `read_lines(20000)` throws an error.

Task 2.4

Write a `count_lines` function that reads a file given as parameter, and prints its total of line(s).

Test your function with both `zen.txt` and `primes.txt` files. To see if it works fine, check each total of lines via a text editor or the *wordcount* command.

Task 2.5

Write a `create_file` function that creates a file named `toto.txt`, in the current directory.

Task 2.6

Write a `write` function that adds the string "I'm a new line" at the end of `toto.txt`.

Task 2.7

Write a `rewrite` function that overwrites the content of `toto.txt` with the content from `zen.txt`.

Task 2.8

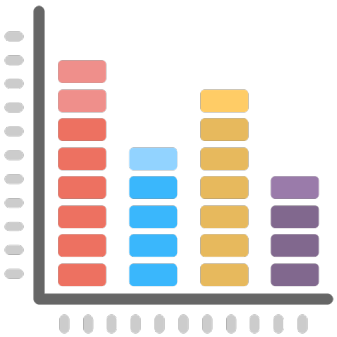
Write a `longest` function that prints the longest word contained in `zen.txt`.

Task 2.9

Write a `frequency` function that counts the frequency of each word contained in `zen.txt`.

Task 2.10

Write a `frequency` function that counts the frequency of each letter contained in `zen.txt`.



v 1.2