Table of Contents

Code Bytes Daily Challenge Explanation (Python)

Robertus Diawan Chris

1. Day 1 - Reverse String

The challenge for day one is reverse string from an input.

On this challenge i use slice string to reverse string (to print the string text backward). Slice string have this syntax `[start:stop:step]` and giving no value as start and stop indicates default to 0 as start and string length as stop, and also if we assign step to "-1" denotes starting from the very end.

So for this challenge the main code is `string[::-1]`. I solve this challenge using external file as an input, the code is below:

```
f = open('input', 'r')

string = f.readlines()

for s in string:
    print(s[::-1])
```

here's inside input file:

Cat
The Daily Byte
civic

and using readlines method we can read each sentence in different lines.

2. Day 2 - Valid Palindrome

The challenge for day two is palindrome case. It's basically check if a sentence read the same forward and backward.

For this challenge i solve it by comparing the actual string and the reverse string version, if it's the same then it's polindrome. I turn all the string into lowercase and remove all symbols before comparing actual string and reverse string. Here is the code:

```
from fire import Fire

def main(input):
    input_without_symbols = input.replace(',', '').replace(':',
'').replace(' ', '').replace('.', '')

    print(f'{input} -> true') if input_without_symbols.lower() ==
input_without_symbols.lower()[::-1] else print(f'{input} -> false')

if __name__ == '__main__':
    Fire(main)
```

I use fire python module to give an input in command line interface, you could give the list directly inside the code if you want. I'm still learning about regular expression so i'm not sure how to implement regex in this case for now.

3. Day 3 - Vacuum Cleaner Route

The challenge for day three is how to determine if a robot vacuum cleaner return to the original position.

U -> Up
D -> Down
L -> Left
R -> Right

So first thing first we need to know that to back to original position we need to define the variable of the moves and the oposite of the move, for example, the oposite of up is down so if the robot going up then the we're gonna add the value of up-down variable and if the robot going down then we're gonna minus the value of up-down variable so if the robot back to original position then the value of up-down varible should be 0. With that in mind, here's the code:

```
from fire import Fire

def main(moves):
    UD = 0 #up and down variable
    LR = 0 #left and right variable

    for m in moves:
        if m == 'U':
            UD += 1
        elif m == 'D':
            UD -= 1
        elif m == 'L':
            LR += 1
        elif m == 'R':
            LR -= 1

    print(UD == 0 and LR == 0)

if __name__ == '__main__':
    Fire(main)
```

4. Day 4 - Correct Capitalization

Given a string, return whether or not it uses capitalization correctly. A string correctly uses capitalization if all letters are capitalized, no letters are capitalized, or only the first letter is capitalized.

Example:

"USA"	return	True
"Nganu"	return	True
"compUter"	return	False
"coding"	return	True

To solve that challenge i use python built-in string methods that is `string.isupper()` to check if all the letter is uppercase and `string.islower()` to check if all the letter is lowercase and for only the first letter is capitalized i compare the result of python built-in string methods that is `string.capitalize()` with the actual word.

With that in mind, here's the code:

```
from fire import Fire

def main(word):
    print(word.isupper() or word.islower() or word == word.capital-
ize())

if __name__ == '__main__':
    Fire(main)
```

5. Day 5 - Add Binary

Given two binary strings (strings containing only 0s and 1s) and then return their sum as a binary strings too. Neither binary string will start with 0s unless the string itself is 0.

Example:

```
"100"    +    "1"    return    "101"
 "11"    +    "1"    return    "100"
  "1"    +    "0"    return     "1"
```

To solve that challenge i use python built-in `bin()` and `int()` function. `bin()` function convert integer number to a binary string prefixed with "0b" and `int()` function return integer object constructed from a number or string given numeric base.

There're two ways you can use `int()` function. That is, using the function with number or with string. Here's the general syntax:

```
int(x)
int(x, base)
```

here's the example: `int(10)` or `int('10', 2)`. Here's a brief explanation about **base** in `int()` function:

> Base 1 only have "0" as a symbol so the lowest base is 2, while base 0 return the string number as it is. Base 2 uses symbols "01". For example, if we do `print('11', 0)` then the result gonna be 11, the actual number. But if we do `print('11', 2)` then the result gonna be 3, the number from the binary string '11'. So if you do `print(bin(3))` then you're gonna get "0b11" which "0b" is the binary indicator (to make it simpler) and 11 is the actual binary.
>
> While the highest base is 36 because it use symbols from "0123456789abcdefghijklmnopqrstuvwxyz" (10 digits + 26 characters, and you could continue with more symbols too).

Below is the table to make it easier to understand:

Base	
0	Print number as it is
2	The lowest base (use "01" symbols)
36	The highest base (use "0123456789abcdefghijklmnopqrstuvwxyz" symbols)

Here's the example for manually calculate using base:

`int('11', 2)` -> 1*(2**1) + 1*(2**0)

here's another example using base-10:

`int('11', 10)` -> 1*(10**1) + 1*(10**0)

With that in mind, here's the code:

```
from fire import Fire

def main(string1, string2):
    a = str(string1)
    b = str(string2)
    s = bin(int(a, 2) + int(b, 2))
    print(s[2::])

if __name__ == '__main__':
    Fire(main)
```

I convert the input to string because fire python module automatically convert the python object even though i use "" to declare as string and it still recognized as integer. For s[2::] means that it start from second part (so 0 and 1 doesn't shown), if i didn't do that then the result gonna be 0b100 which 100 is the binary of the result and we didn't really need 0b.