# CS50P - Problem Set 4

## Emojize

* Implement a program that prompts the user for a **str** in English and then outputs the “emojized” version of that str.
* Converting any codes (:thumbs\_up:) or aliases (:thumbsup:) to emoji
* **from emoji** **import** emojize
* s\_parts = s.split(“ “)

result = [ ]

* **for** part **in** parts:

**if** “**:**” **in** part **and** “**\_**” in part:

result.append(emojize(part, language=”alias”))

* **return** **f**”Output: {“ “.join(result)}”

## Frank, Ian and Glen’s Letters

* Implement a program that **zero** or **two** command-line arguments:
  + Zero if the user would like to output text in random font.
  + Two if the user would like to output text in a **specific font**, in which case the first of the two should be **-f** or **–font**, and the second of the two should be the **name of the font**.
* Prompts the user for a str of text
* Outputs that text in desired font
* If the user provides two command-line arguments and the first is not -f or –font or the second is not the name of a font, the program should **exit via sys.exit with an error message.**
* import sys

import random

form pyfiglet import Figlet

* **if** len(sys.argv) == n:

…

* sys.exit(“Invalid usage”)
* **if** sys.argv[1] == “-f” **or** sys.argv[1] == “--font” **and** sys.argv[2] **in** fonts:

…

## Adieu, Adieu

* Implement a program that prompts the user for names, **one per line**, until the user inputs **control-d**.
* Assume that the user will input **at least one name**.
* Bid adieu to those names, separating two names with **one and**
* Three names with **two commas** and **one and**
* And n names with **n-1 commas** and **one and**
* “Adieu, adieu, to Liesl”
* “Adieu, adieu, to Liesl, and Friedrich”
* “Adieu, adieu, to Liesl, Friedrich, and Louisa”
* “Adieu, adieu, to Liesl, Friedrich, Louisa, and Kurt”
* **Import** inflect

p = inflect.engine()

names = [ ]

* **try:**

name = input(“Name: ”)

names.append(name)

* **except** EOFError:
* print(f“Adieu, adieu to **{**p.join(names, final\_sep=”**,**”)**}**”)

## Guessing Game

* Prompt user for a level, **n**.
* If the user does not input a **positive integer**, the program should prompt again.
* Randomly generates an integer between **1** and **n**, inclusive, using the **random** module.
* Prompt the user to guess that integer
  + If the guess is not positive integer - **prompt again**
  + If the guess is smaller than that integer - output “Too small!”
  + If the guess is larger than that integer - output “Too large!”
  + If the guess is the same as that integer - output “Just right!”
* **import** random

**import** sys

* random.randint(**1**, n)
* **except** ValueError
* sys.exit()

## Little Professor

* Prompt the user for a level **n**
  + If the user does not input **1,2 or 3,** prompt again
* Randomly generate **10 math problems** formatted **X + Y =** . Wherein each of X and Y is a **non-negative** integer with **n** digits.
* No need to support other operations other than addition (+)
* **Prompt the user to solve each of those problems :**
  + If an answer is **not correct** (or **not an integer**), the program should **output EEE** and **prompt the user again**
  + Allow the user up to three tries in total for that problem.
  + If the user still has **not answered correctly after three tries**, **output the correct answer**.
* The program should **ultimately output the user’s score**: **The number of correct answers out of 10.**
* **get\_level()** to prompt (and **if need be re-prompt**) the user for a level and return 1, 2 or 3
* **generate\_integer()** returns randomly generated non-negative integer with level digits or raises a **ValueError if level is not 1, 2 or 3**
* **import** random
* get\_level():
* **def** generate integer(level):

**if** level == X:

**return** random.randint(0,9)

…

* **for** i **in** range(10):

**for** j **in** range(3):

answer = int(intput(**f**”{int\_1} + {int\_2} = ”)

**if** isinstance(answer, int) and answer == solution:

score += 1

break

**else**:

print(“EEE”)

**if** j == 2:

print(**f**”{int\_1} + {int\_2} = {solution} ”)

i **+=** 1

* **except** ValueError

## Bitcoin Price Index

Implement a program that:

- Expects the user to specify as a **command-line argument** the number of Bitcoins, **n**, that they would like to buy.

- If that argument cannot be converted to a **float**, the program should exit via **sys.exit** with an error message.

- Queries the API for the CoinDesk Bitcoin Price Index at <https://api.coindesk.com/v1/bpi/currentprice.json> which returns a JSON object, among whose nested keys is the current price of Bitcoin as a float.

- Be sure to catch any exceptions

- Output the current **cost of n Bitcoins** in **USD** to **four decimal places**, using “**,**” as a thousands separator

* **import** json

**import** requests

**import** sys

* r = requests.get(“<https://api.coindesk.com/v1/bpi/currentprice.json>”)

r\_json = r.json()

**return** r\_json[“bpi”][“USD”][“rate\_float”]

\*To access a value inside a dictionary inside another inside another dictionary: [“key1”][“key2”][“key3”]

* **except** requests.RequestException:

sys.exit(“Error with file”)

* bitcoins = float(sys.argv[1])
* **if** \_\_name\_\_ == “\_\_main\_\_”:

main()