# CS50P Problem Set 5

## Testing my twttr

1. Reimplement **`twttr.py`** from problem 2, restructuring the code:

* **shorten()** expects a **str** as input and returns that same str but with all vowels (**A, E, I, O, and U) omitted**. Whether inputted in uppercase or lowercase.

1. Then, in a file called `**test\_twttr.py**` implement one or more functions that collectively test your implementation of shorten thoroughly.

* Each of whose names should begin with `**test\_**`
* Execute your tests with : `**pytest test\_twttr.py**`

### twttr.py

def main():

word = input("Input: ").strip()

print(shorten(word))

def shorten(word):

vowels = ["A", "a", "E", "e", "I", "i", "O", "o", "U", "u"]

new\_word\_list = [ ]

**for** c **in** word:

**if** c **not** vowels:

new\_word\_list.append(c)

new\_word = **“”**.join(new\_word\_list)

return new\_word

if \_\_name\_\_ == "\_\_main\_\_":

main()

### 

### test\_twttr.py

import **pytest**

from **twttr** import **shorten**

def test\_alpha\_lower():

assert shorten(“twitter”) == “twttr”

assert shorten(“lazer”) == “lzr”

def test\_alpha\_upper():

assert shorten(“GOOGLECHROME”) == “GGLCHRM”

assert shorten(“TEST”) == “TST”

def test\_alphanum():

assert shorten(“cs50”) == “cs50”

assert shorten(“Hai18”) == “H18”

def test\_punctuation():

assert shorten(“Bar.Yohai”) == “Br.Yh”

assert shorten(“Hallo!”) == “Hll!”

## Back to the Bank

1. Reimplement **Home Federal Saving Bank** from Problem Set 1 `**bank.py**`:

* **value()** expects a **str** as input and **returns an int**
  + Namely **0** if that str **starts with “hello**”, **20** if it **starts with an “h”**, or **100** otherwise
  + Treat the str **case-insensitively**.
* Assume that the string passed to **value()** will not contain leading spaces
* Only main should call print

1. Then, in a file called `**test\_bank.py**` implement **3** or more functions that collectively test your implementation of value() thoroughly.
   * Each of whose names should begin with `**test\_**`
   * Execute your tests with : `**pytest test\_bank.py**`

### bank.py

def main():

while True:

greeting = input("Greeting: ")

if not greeting:

raise ValueError(“Please enter greeting”)

print(**f**”$**{**value(greeting)**}**”)

break

def value(greeting):

if greeting.startswith((“hello”, “HELLO”, “Hello”)):

return 0

elif greeting.startswith((“h”, “H”)):

return 20

else:

return 100

if \_\_name\_\_ == "\_\_main\_\_":

main()

### test\_bank.py

import **pytest**

from **bank** import **value**

def test\_lower():

assert value(“hello patrick”) == 0

assert value(“herrow”) == 20

assert value(“skamlikum”) == 100

def test\_upper():

assert value(“HELLO BRO”) == 0

assert value(“HERROW MALFOY”) == 20

assert value(“S’UP”) == 100

def test\_up\_low():

assert value(“Hello!”) == **0**

assert value(“hErrOW jUdE”) == 20

assert value(“Ello Bro”) == 100

## Re-requesting a Vanity Plate

1. Reimplement Vanity Plate from Problem Set 2 `**plates.py**`

* **is\_valid()** expects a **str** as input
  + returns **True** if that str meets all requirements.
  + returns **False** if it does not.
  + **main()** only called if \_\_name\_\_ == “”\_\_main\_\_

1. Then, in a file called `**test\_plates.py**` implement **4** or more functions that collectively test your implementation of **is\_valid()** thoroughly.
   * Each of whose names should begin with `**test\_**`
   * Execute your tests with : `**pytest test\_plates.py**`

### plates.py

def main():

plate = input("Plate: ").strip()

if is\_valid(plate):

print("Valid")

else:

print("Invalid")

def is\_valid(s):

zero\_digit\_found = False

if len(s) < 2 or len(s) > 6 or s[0].isalpha() == False

or s[1].isalpha() == False:

return False

for i, c in enumerate(s):

if c.isalnum() == False:

return False

if c.isdigit():

if c == "0" and zero\_digit\_found == False:

return False

elif c != "0":

zero\_digit\_found = True

if c.isalpha() and i != 0 and s[i - 1].isdigit():

return False

return True

if \_\_name\_\_ == "\_\_main\_\_":

main()

### 

### test\_plates.py

import pytest

from plates import is\_valid

def test\_starts\_2\_letters():

assert is\_valid("HELLO") == True

assert is\_valid("AE332") == True

assert is\_valid("A3356") == False

assert is\_valid("5566O") == False

def test\_min\_2\_characters():

assert is\_valid("H") == False

assert is\_valid("AA") == True

assert is\_valid("5") == False

assert is\_valid("AA5") == True

def test\_max\_6\_letters():

assert is\_valid("LAZERY") == True

assert is\_valid("GOODBYE") == False

assert is\_valid("EMET26") == True

assert is\_valid("OK84567") == False

def test\_number\_placement():

assert is\_valid("AA1826") == True

assert is\_valid("AAA222") == True

assert is\_valid("CS50P") == False

assert is\_valid("ZZ44TP") == False

def test\_zero\_placement():

assert is\_valid("CS50") == True

assert is\_valid("CS05") == False

def test\_special\_characters():

assert is\_valid("HELLO, WORLD") == False

assert is\_valid("PI3.14") == False

assert is\_valid("YO700!") == False

## Refueling

1. Reimplement **Fuel Gauge** from Problem Set 2 `**fuel.py**`

* **convert()** expects a **str** in X/Y format input
  + Each of X and Y is an **integer**
  + Returns that fraction as a **percentage** rounded to the nearest **int** between **0** and **100** inclusive.
  + If X and/or Y is not an integer, or if X is greater than Y, then convert should raise a ValueError.
  + If Y is 0 then convert should raise a ZeroDivisionError.
* **gauge()** expects an int and returns a str that is:
  + **“E”** if that int is less than or equal to **1**
  + **“F”** if that int is greater than or equal to **99**
  + “**Z%**” otherwise, where Z is that same **int**

1. Then, in a file called `**test\_fuel.py**` implement **2** or more functions that collectively test your implementation of **convert()** and **gauge()**  thoroughly.
   * Each of whose names should begin with `**test\_**`
   * Execute your tests with : `**pytest test\_plates.py**`

### fuel.py

def main():

fraction = input("Fraction: ")

percentage = convert(fraction)

print(gauge(percentage))

def convert(fraction):

try:

x, y = fraction.split("/")

x = int(x)

y = int(y)

if y == 0:

raise ZeroDivisionError("Denominator should be greater than 0")

if x > y:

raise ValueError("Numerator should not be greater than denominator")

except ValueError:

raise ValueError("Please enter integer values")

return round((float(x / y) \* 100))

def gauge(percentage):

if percentage <= 1:

return "E"

elif percentage >= 99:

return "F"

else:

return str(percentage) + "%"

if \_\_name\_\_ == "\_\_main\_\_":

main()

### test\_fuel.py

import pytest

from fuel import convert, gauge

def test\_convert():

assert convert("3/4") == 75

assert convert("1/4") == 25

def test\_convert\_exceptions():

with pytest.raises(ZeroDivisionError):

convert("4/0")

with pytest.raises(ValueError):

convert("three/four")

with pytest.raises(ValueError):

convert("1.5/3")

def test\_gauge():

assert gauge(1) == "E"

assert gauge(99) == "F"

assert gauge(25) == "25%"

assert gauge(75) == "75%"