Patient Class

In this exercise we will make an "Patient" class. The Patient class should store the state of a patient in our hospital system.

Problem 1:

Create a class called "Patient". The constructor should have two parameters (in addition to self, of course):

1. name (str)
2. symptoms (list of str)

the parameters should be stored as attributes called "name" and "symptoms" respectively

In [1]:
```
class Patient:
    def __init__ (self, name, symptoms):
        self.name = name
        self.symptoms = symptoms

patient_1 = Patient("Silvia", ["fever", "cough", "anosmia"])
patient_2 = Patient("Marta", ["sorethroat", "headache", "fever"])
```

In [2]:
```
patient_1.name
```

Out[2]:
```
'Silvia'
```

In [3]:
```
patient_1.symptoms
```

Out[3]:
```
['fever', 'cough', 'anosmia']
```

Problem 2:

Create a method called "add_test" which takes two paramters:

1. the name of the test (str)
2. the results of the test (bool)

This information should be stored somehow.

In [4]:
```
class Patient:
    def __init__ (self, name, symptoms):
        self.name = name
        self.symptoms = symptoms
        self.tests_results = []

    def add_test (self, test, result):
        test_result_pair = (test, result)
        self.tests_results.append(test_result_pair)
```

In [37]:
```
# version 2 (with dictionaries)
class Patient:
    def __init__ (self, name, symptoms):
        self.name = name
        self.symptoms = symptoms
        self.tests_results = {}

    def add_test (self, test, result):
        self.tests_results[test] = result
```

In [49]:
```
patient_2.tests_results['COVID']
```

Out[49]:
```
False
```

In [29]:
```
patient_1.symptoms
patient_1.add_test('Hernia', False)
patient_1.tests_results
```

Out[29]:
```
[('Hernia', False)]
```

In [39]:
```
patient_2 = Patient('Erika', ['fever', 'cough', 'anosmia'])
patient_2.add_test('COVID', False)
patient_2.add_test('Herpes', True)
```

Problem 3:

Create a method called has_covid() which takes no parameters.

"has_covid" returns a float, between 0.0 and 1.0, which represents the probability of the patient to have Covid-19

The probability should work as follows:

If the user has had the test "covid" then it should return .99 if the test is True and 0.01 if the test is False Otherwise, probability starts at 0.05 and ncreases by 0.1 for each of the following symptoms: ['fever', 'cough', 'anosmia']

In [7]:
```
# version 1: Tuples
class Patient:
    def __init__ (self, name, symptoms):
        self.name = name
        self.symptoms = symptoms
        self.tests_results = []
        self.probability_covid = 0

    def add_test (self, test, result):
        test_result_pair = (test, result)
        self.tests_results.append(test_result_pair)

    def has_covid(self):
        probability_covid = 0.05
        covid_symptoms = ['fever', 'cough', 'anosmia']

        if not self.tests_results:

             for i in self.symptoms:
                    if i in covid_symptoms:
                        probability_covid += 0.1

        for i in range(len(self.tests_results)):
            if self.tests_results[i][0] == 'COVID':
                if self.tests_results[i][1] == True:
                    probability_covid = 0.99
                if self.tests_results[i][1] == False:
                    probability_covid = 0.01


        self.probability_covid  = probability_covid
        return probability_covid
```

In [9]:
```
# version 2 (with dictionaries)

class Patient:
    def __init__ (self, name, symptoms):
        self.name = name
        self.symptoms = symptoms
        self.tests_results = {}
        self.probability_covid = 0

    def add_test (self, test, result):
        self.tests_results[test] = result

    def has_covid(self):
        probability_covid = 0.05
        covid_symptoms = ['fever', 'cough', 'anosmia']

        try:
            if self.tests_results['COVID']:
                probability_covid = 0.99

            elif self.tests_results['COVID'] == False:
                probability_covid = 0.01

        except:
            for i in self.symptoms:
                if i in covid_symptoms:
                        probability_covid += 0.1
        self.probability_covid  = probability_covid
        return probability_covid
```

In [11]:
```
patient_7 = Patient('Erika', ['fever', 'cough', 'anosmia'])
patient_7.add_test('Herpes', False)
```

In [12]:
```
patient_7.has_covid()
```

Out[12]:
```
0.35
```

In [60]:
```
patient_5.add_test('COVID', True)
patient_5.__dict__
```

Out[60]:
```
{'name': 'Maggie',
 'symptoms': ['knee pain', 'nightmares', 'fever', 'cough', 'anosmia'],
 'tests_results': {'COVID': True},
 'probability_covid': 0}
```

In [10]:
```
patient_5.has_covid()
```

Out[10]:
```
0.01
```

Encoder Class

In this exercise we will make an "Encoder" class

The Encoder class should be able to encode a list of strings into a list of integers that can later be losslessly decoded.

For example, if given a list of words: ['Joan', 'went', 'to', 'the', 'store'] it might encode: [245, 9873, 290, 10, 209] and be able to decode it back again to the list of words.

HINT: you can use an integer attribute called "index"; whenever you encounter a new word (string), increment the index and use that value to encode the word

Problem 4:

Create a class called "Encoder." The constructor should have no parameters (besides, of course, "self")

In [ ]:
```
class Encoder:
    def __init__ (self):
```

Problem 5:

Add two methods: "encode" and "decode"

"encode" should have a single parameter, a list of strings, and returns a list of integers which represents the encoding.

"decode" should have a single parameter, a list of integers, and returns a list of strings, which should be the same as was passed to "encode"

In [18]:
```
class Encoder:
    def __init__(self):
        self.word_number_list = []  # Initialize an empty list to store word-number pairs (tuples)

    def encode(self, word_list):
        number_list = []  # Initialize an empty list to store the encoded numbers
        for index, word in enumerate(word_list):  # Iterate over the word_list while keeping track of the index
            self.word_number_list.append((word, index))  # Append the word-number tuple to the list
            number_list.append(index)  # Append the index to the number_list

        return number_list

    def decode(self, number_list):
        back_to_words = []  # Initialize an empty list to store the decoded words
        for number in number_list:  # Iterate over the numbers in the number_list
            for word, num in self.word_number_list:  # Iterate over the word-number tuples in the list
                if num == number:  # Check if the number matches the index in the tuple
                    back_to_words.append(word)  # Append the word to the back_to_words list
        return back_to_words
```

In [23]:
```
the_encoder = Encoder()
```

In [24]:
```
the_word_list = ['Joan', 'went', 'to', 'the', 'store']
```

In [25]:
```
the_encoded_list = the_encoder.encode(the_word_list)
print(the_encoded_list)
```

```
[0, 1, 2, 3, 4]
```

In [26]:
```
the_encoder.__dict__
```

Out[26]:
```
{'word_number_list': [('Joan', 0),
  ('went', 1),
  ('to', 2),
  ('the', 3),
  ('store', 4)]}
```

In [16]:
```
the_number_list = [4, 2]
the_decoded_list = the_encoder.decode(the_number_list)
print(the_decoded_list)
```

```
['store', 'to']
```

In [ ]: