Homework 9

Due Nov 2nd, 2020, 4PM

In [166]:
```
first_name = "Shariq"
last_name = "Jamil"

assert(len(first_name) != 0)
assert(len(last_name) != 0)
```

1) Point

Modify class Point defined below to provide working versions of **str**() and **eq**().

Edit the class so that two Points with the same x and y are the same, and so that points are printed as tuples.

Printing

```
one = Point(3, 4)
print(one)
```

Should produce:

```
(3, 4)
```

Double Equals

```
one = Point(3, 4)
two = Point(3, 4)
print(one == two)
```

Should produce:

```
True
```

In [167]:
```
class Point(object):
    """Represents a point in 2-D space."""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        # return x and y as a string in tuple format
```

```
        return '(' + str(self.x) + ', ' + str(self.y) + ')'

    def __eq__(self, other):
        # compare x and y for two points
        # return True if they are equal, False if not
        return self.x == other.x and self.y == other.y
```

Unit Test for Point

In [168]:
```
def test_point():
    p = Point(3, 4)
    q = Point(3, 4)

    assert p.__str__() == '(3, 4)', "Should yield (3 4)"
    assert p == q, "Should be equal"

    print('Success!')

test_point()
```

```
Success!
```

2) Collatz sequence

The Collatz sequence, also know as the Hailstone sequence, is a sequence of numbers.

If the current number is n, the next number is n / 2 if n is even, and 3n + 1 if n is odd.

It has not been shown that there isn't a sequence which never repeats.
All known sequences end by repeating 4, 2, 1, 4, 2, 1, ...

Write a generator collatz(n) **that starts at n** and generates the rest of the sequence down to 1.
Your generator should raise a StopIteration exception after yielding 1.

In [169]:
```
def collatz(n):
    # yield starting n
    yield n
    while True:
        # finish processing if n is 1
        if n == 1:
            return
        # if n is odd
        if n % 2:
            # next number is 3n + 1
            n = 3*n+1
            yield n
        else:
            # next number is n/2
            n = n//2
            yield n
```

Unit Tests

In [170]:
```
def test_collatz():
    g = collatz(4)
    lst = [n for n in g]
    assert lst == [4, 2, 1]

    g = collatz(11)
    lst = [n for n in g]
    assert lst == [11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2,

    g = collatz(29)
    lst = [n for n in g]
    assert lst == [29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10,

    print('Success!')

test_collatz()
```

```
Success!
```

3) Next Month

Write a generator that will return a sequence of month names. Thus

```
gen = next_month('October')
```

creates a generator that generates the strings 'November', 'December', 'January' and so on. If the caller supplies an illegal month name, your function should raise a ValueError exception with text explaining the problem.

In [171]:
```
month_names = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November',

def next_month(name: str) -> str:
    global month_names

    # get index of given month
    month_index = month_names.index(name.capitalize())
    # check if exists
    if month_index != -1:
        while True:
            # increment month
            month_index += 1
            # if at December
            if month_index == 12:
                # loop back to the beginning
                month_index = 0
            yield month_names[month_index]
    # unrecognized month
    else:
        raise ValueError("Please provide a valid month")
```

Unit Tests

In [172]:
```
def test_months():
    gen = next_month('October')
    lst = [next(gen) for i in range(15)]
    assert(lst == ['November', 'December', 'January', 'February', 'Mar

    gen = next_month('december')
    assert next(gen) == 'January'

test_months()
```

The following should raise a ValueError with text explaining the problem

In [173]:
```
try: # This should throw a Value Error
    gen = next_month('Thermador')

    m = next(gen)

    print(1/0)
except ValueError:
    print('Success!')
```

```
Success!
```

4) Phone Numbers

Modify the class below that takes a string and returns an object holding a valid NANP phone number. You will need to fill in the three methods listed, but underfined, below: __str__(), area_code(), and normalize().

The "North American Numbering Plan" (NANP) is a telephone numbering system used by many countries in North America. All NANP-countries share the same international country code: `1` .

NANP numbers are ten-digit numbers consisting of a three-digit area code and a seven-digit local number. The first three digits of the local number are the "exchange code", and the four-digit number which follows is the "subscriber number".

The format is usually represented as (NXX)-NXX-XXXX where `N` is any digit from 2 through 9 and `X` is any digit from 0 through 9.

Your task is to clean up differently formatted telephone numbers by removing punctuation, such as '(', '-', and the like, and removing and the country code (1) if present.

Start by stripping non-digits, and then see if the digits match the pattern. If you are asked to create a phone number that does not meet the pattern above, you should throw a ValueError.

with a string explaining the problem: too many or too few digits, or the wrong digits.

For example, the strings below

+1 (617) 495-4024

617-495-4024

1 617 495 4024

617.495.4024

should all produce an object that is printed as (617) 495-4024

ValueErrors

Each of the following strings should produce a ValueError exception.

+1 (617) 495-40247 has too many digits

(617) 495-402 has too few digits

+2 (617) 495-4024 has the wrong country code

(017) 495-4024 has an illegal area code

(617) 195-4024 has an illegal exchange code

In [181]:
```
class Phone:
    "A Class defining valid Phone Numbers"

    def __init__(self, raw):
        "Create new instance"
        self.number = self._normalize(raw)

    def __str__(self) -> str:
        return self.number

    def area_code(self) -> str:
        "Return the area code"
        return self.number[1:4]

    def _normalize(self, raw: str) -> str:
        """Take string presented and return string with digits
            Throws a ValueError Exception if not a NANP number"""
        phone_number = ''
        for char in raw:
            # remove non-numeric chars
            if char.isdigit():
                phone_number += char
        if len(phone_number) == 11:
            if phone_number[0] == '1':
                # remove country code
                phone_number = phone_number[1:]
            else:
```

```
            raise ValueError('Invalid phone number: Wrong country
        if len(phone_number) > 10:
            raise ValueError('Invalid phone number: Too many digits')
        elif len(phone_number) < 10:
            raise ValueError('Invalid phone number: Too few digits')
        # extract into NANP segments
        area_code = phone_number[0:3]
        # verify that the first digit of area code is between 2 and 9
        if int(area_code[0]) < 2:
            raise ValueError('Invalid phone number: Illegal area code'
        exchange_code = phone_number[3:6]
        # verify that the first digit of exchange code is between 2 an
        if int(exchange_code[0]) < 2:
            raise ValueError('Invalid phone number: Illegal exchange c
        subscriber_number = phone_number[6:10]
        # return NXX-NXX-XXX as (NXX) NXX-XXXX
        return '(' + area_code + ') ' + exchange_code + '-' + subscri
```

Unit Tests for Phone Number

In [182]:
```
def test_phone():
    p = Phone('+1 (617) 495-4024')
    assert(p.__str__() == '(617) 495-4024')

    p = Phone('617-495-4024')
    assert(p.__str__() == '(617) 495-4024')

    p = Phone('1 617 495 4024')
    assert(p.__str__() == '(617) 495-4024')

    p = Phone('617.495.4024')
    assert(p.__str__() == '(617) 495-4024')
    assert(p.area_code() == '617')


    p = Phone('+1 (508) 495  4024')
    assert(p.__str__() == '(508) 495-4024')

    p = Phone('508 - 495 - 4024')
    assert(p.__str__() == '(508) 495-4024')

    p = Phone('1 508 (495) [4024]')
    assert(p.__str__() == '(508) 495-4024')

    p = Phone('508!495?4024')
    assert(p.__str__() == '(508) 495-4024')
    assert(p.area_code() == '508')

    p = Phone('5084950000')
    assert(p.__str__() == '(508) 495-0000')


    print("Success!")

test_phone()
```

```
Success!
```

Unit Tests for invalid numbers - each should raise a ValueError with a different string

There are 5 different problems below: you should throw Value errors with 5 different explainations

In [183]:
```
p = Phone('+1 (617) 495-40247')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-183-db4ce11c1cff> in <module>
----> 1 p = Phone('+1 (617) 495-40247')

<ipython-input-181-e132b87a01ff> in __init__(self, raw)
      4     def __init__(self, raw):
      5         "Create new instance"
----> 6         self.number = self._normalize(raw)
      7
      8     def __str__(self) -> str:

<ipython-input-181-e132b87a01ff> in _normalize(self, raw)
     28                 raise ValueError('Invalid phone number: Wrong country code')
     29         if len(phone_number) > 10:
```

In [184]:
```
p = Phone('(617) 495-402')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-184-692bc6319a88> in <module>
----> 1 p = Phone('(617) 495-402')

<ipython-input-181-e132b87a01ff> in __init__(self, raw)
      4     def __init__(self, raw):
      5         "Create new instance"
----> 6         self.number = self._normalize(raw)
      7
      8     def __str__(self) -> str:

<ipython-input-181-e132b87a01ff> in _normalize(self, raw)
     30             raise ValueError('Invalid phone number: Too many digits')
     31         elif len(phone_number) < 10:
---> 32             raise ValueError('Invalid phone number: Too few digits')
     33         # extract into NANP segments
     34         area_code = phone_number[0:3]

ValueError: Invalid phone number: Too few digits
```

In [185]:
```
p = Phone('+2 (617) 495-4024')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-185-233260c5c685> in <module>
----> 1 p = Phone('+2 (617) 495-4024')

<ipython-input-181-e132b87a01ff> in __init__(self, raw)
      4     def __init__(self, raw):
      5         "Create new instance"
----> 6         self.number = self._normalize(raw)
      7
      8     def __str__(self) -> str:

<ipython-input-181-e132b87a01ff> in _normalize(self, raw)
     26                 phone_number = phone_number[1:]
     27             else:
---> 28                 raise ValueError('Invalid phone number: Wrong
```

In [186]:
```
p = Phone('(017) 495-4024')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-186-225915713b41> in <module>
----> 1 p = Phone('(017) 495-4024')

<ipython-input-181-e132b87a01ff> in __init__(self, raw)
      4     def __init__(self, raw):
      5         "Create new instance"
----> 6         self.number = self._normalize(raw)
      7
      8     def __str__(self) -> str:

<ipython-input-181-e132b87a01ff> in _normalize(self, raw)
     35         # verify that the first digit of area code is between 2 and 9
     36         if int(area_code[0]) < 2:
---> 37             raise ValueError('Invalid phone number: Illegal area code')
     38         exchange_code = phone_number[3:6]
     39         # verify that the first digit of exchange code is between 2 and 9

ValueError: Invalid phone number: Illegal area code
```

In [187]:
```
p = Phone('(617) 195-4024')
```

```
---------------------------------------------------------------------------
------
ValueError                                Traceback (most recent call
last)
<ipython-input-187-895781462c15> in <module>
----> 1 p = Phone('(617) 195-4024')

<ipython-input-181-e132b87a01ff> in __init__(self, raw)
      4     def __init__(self, raw):
      5         "Create new instance"
----> 6         self.number = self._normalize(raw)
      7
      8     def __str__(self) -> str:

<ipython-input-181-e132b87a01ff> in _normalize(self, raw)
     39         # verify that the first digit of exchange code is bet
ween 2 and 9
     40         if int(exchange_code[0]) < 2:
---> 41             raise ValueError('Invalid phone number: Illegal e
xchange code')
```