Homework 8

Due October 26

Fill in your name

In [285]:
```
first_name = "Shariq"
last_name = "Jamil"

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

Problem 1: Odds and evens

Write a function that uses a List Comprehensions to split a list of integers into two halves: the odds, followed by the evens. Within each half, the numbers should appear in their original order.

In [272]:
```
'''
This function takes in a list of numbers
and returns a list with the odd numbers followed by
the even numbers in the order that they appear in
the given list
'''
def odds_n_evens(lst):
    # collect odd numbers from given list
    odd_list =  [x for x in lst if not x%2 == 0]
    # collect even numbers from given list
    even_list =  [x for x in lst if x%2 == 0]
    # create a return list with the odd numbers list before the even n
    res = odd_list+even_list
```

Unit Tests

In [273]:
```
def test_odd_n_even():
    assert odds_n_evens([1, 2, 3, 4]) == [1, 3, 2, 4]
    assert odds_n_evens([4, 3, 2, 1]) == [3, 1, 4, 2]
    assert odds_n_evens([]) == []
    assert odds_n_evens([1, 3, 5, 7, 9]) == [1, 3, 5, 7, 9]
    assert odds_n_evens([9, 7, 5, 3, 1]) == [9, 7, 5, 3, 1]
    assert odds_n_evens([2, 4, 6, 8]) == [2, 4, 6, 8]
    assert odds_n_evens([1, 2, 3, 4, 1, 2, 3, 4]) == [1, 3, 1, 3, 2, 4

    print("Success!")
```

```
Success!
```

Problem 2: DNA Complement

In a DNA sequence, the symbols 'A' and 'T' are complements of each other, as are 'C' and 'G'.

The complement of a DNA sequence is the string formed by reversing the sequence, and then taking the complement of each symbol

Write a function that takes a string representing a DNA sequence, and returns the a string representing the complement.

Hint: Use a Dictionary to map a symbol ('A') to its complement ('T')

Hint: for full credit, use join() rather than string addition

The type hint for this function would be

```
def dna_complement(text: str) -> str:
```

In [274]:
```
def dna_complement(text):
    # list of DNA symbol complements
    complements = {
        "A":"T",
        "T":"A",
        "C":"G",
        "G":"C",
    }
    # build a list of complements for each character in the given stri
    res = [complements[char.upper()] for char in text]
    # compile the characters after reversing the order of symbols
    str = "".join(reversed(res))
```

Unit Tests

In [275]:
```
def test_complement():
    assert(dna_complement('A') == 'T')
    assert(dna_complement('a') == 'T')
    assert(dna_complement('c') == 'G')
    assert(dna_complement('CaT') == 'ATG')
    assert(dna_complement('AAAACCCGGT') == 'ACCGGGTTTT')
    assert dna_complement('AcgTTTcAgCCC') == 'GGGCTGAAACGT'

    print("Success!")
```

```
Success!
```

Problem 3: Finding the First Link

You can now fetch the contents of a webpage, and had a small taste of its contents

While the copyright appears on almost every webpage, you may have figured out that there is little agreement on how it should appear. It is a bit like musicians royalties today: everyone agrees it is important, but there are no standards, and the only goal seems to be to make it as small as possible.

Links, however, are a different story. They are a key element of the web, and well defined. Here is the syntax for an 'anchor' (aka link)

```
<a href="url">link text</a>
```

To locate a link on a webpage, we can search for the following three things in order:

- First, look for the three character stirng '<a '
- Next, look for the following to close to the first part '>': Those enclose the URL
- Finally, look for three characters to close the element: '</a': which marks the end of the link text

The anchor has two parts we are interested in: the URL, and the link text.

Write a function that takes a URL to a webpage and finds the **first link** on the page. Your function should return a tuple holding two strings: the URL, and the link text

In [276]:

```
# Modified given read_url.py
# by Jeff Parker
#
# This func takes in a website URL and then
# returns the contents of the webpage
#
# Usage:
#     python fetch_webpage.py <url>

import urllib.request
import sys

def fetch_webpage(url):
    try:
        with urllib.request.urlopen(url) as f:
            text = f.read().decode('utf-8')
        return text

    except urllib.error.URLError as e:
        print(e.reason)
        return []

'''
This function takes in a website URL and returns
the first link on the webpage
'''
def find_first_link(url):
    # get webpage as string from URl
    res = fetch_webpage(url)
    start_index = 0
    # find and store the index of the first link
    start_index = res.find('<a')
```

```
    # find the '>' of the opening tag
    close_open_tag = res.find('>', start_index)
    # parse href link and store
    url = (res[start_index+3:close_open_tag])
    # find closing tag
    close_tag = res.find('</a>', start_index)
    # parse link text
    link = (res[close_open_tag+1:close_tag])
    # return as tuple
```

Unit Tests

In [277]:
```
for url in ['http://www.python.org/', 'https://www.extension.harvard.e
    print(f"{url}\n\t{find_first_link(url)}")
```

```
http://www.python.org/ (http://www.python.org/)
        ('href="http://browsehappy.com/"', 'upgrade to a different br
owser')
https://www.extension.harvard.edu (https://www.extension.harvard.edu)
        ('href="#main-menu" class="skip"', 'Jump to navigation')
http://en.wikipedia.org/wiki/Python (http://en.wikipedia.org/wiki/Pyt
hon)
        ('id="top"', '')
Done
```

Problem 4: Dates

Fill in the defintion of the three method below for a class Date

In [278]:
```
class Date(object):
    "Represents a Calendar date"

    def __init__(self, day=0, month=0, year=0):
        # Initialize
        self.day = day
        self.month = month
        self.year = year


    def __str__(self):
        # Format string in the order of month, day, year
        # with '/' the character seperating the units
        return ('%d/%d/%d' % (self.month, self.day, self.year))

    def before(self, other):
        # Returns True if self date is before other date, processed in
        # year, month, day
        return (self.year, self.month, self.day) < (other.year, other.
```

Unit Tests

In [279]:
```
def test_dates():
    t1 = Date(1, 2, 3)
    assert t1.__str__() == '2/1/3'
    t2 = Date(4, 5, 2)
    assert t2.__str__() == '5/4/2'

    assert not t1.before(t1)
    assert not t1.before(t2)
    assert t2.before(t1)

    t2 = Date(4, 1, 3)
    assert t2.__str__() == '1/4/3'

    assert not t1.before(t1)
    assert not t1.before(t2)

    t1 = Date(2, 2, 3)
    t2 = Date(1, 2, 3)
    assert t2.__str__() == '2/1/3'

    assert not t1.before(t1)
    assert not t1.before(t2)
    assert t2.before(t1)

    print("Success!")
```

```
Success!
```

Problem 5: Time after Time

You will not write a lot of code for this problem, but it is a realistic introduction to maintaining a piece of software. Downey's program works, but we want to make two changes.

- Downey prints time as they do in the Army: 17:30:00 hours. We want to print that as 5:30 PM.
- Downey lets you define the time 25:00:00 - we want to turn over at 23:59:59 to 00:00:00.

My advice is to spend more time thinking and tracing out the logic and less time editing.

Make a backup of the cell below, and make your changes

We will want you to identify your changes, so sign everything you do # like this - jdp

Modify Downey's Time2.py file to make the following changes.

A) Rewrite the dunder str method used to print the time. It currently prints Time(17, 30, 0) as

```
17:30:00
```

Modify it to return

```
5:30 PM
```

Hours are numbers between 1 and 12 inclusive, seconds are suppressed, and times end with AM or PM. For purposes of this problem, midnight is AM, while noon is PM.

B) Time2.py currently allows you to create times with hours greater than 23. Identify the routines that Downey provides that would have to change to keep hours less than 24.

C) Make the changes required to keep hours less than 24.

D) Include the tests you have used to verify your changes.

Run the unit tests: all times should be within 24 hours

Make your changes in the cell below

In [317]:

```
"""

Code example from Think Python, by Allen B. Downey.
Available from http://thinkpython.com

Copyright 2012 Allen B. Downey.
Distributed under the GNU General Public License at gnu.org/licenses/g

"""

class Time(object):
    """Represents the time of day.

    attributes: hour, minute, second
    """
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    # Modify this routine - jdp
    def __str__(self):
        am_or_pm = 'AM' # - smj
        # keep hours under 24
        if self.hour > 24: # - smj
            # if greater, process days and find remaining hours
            self.hour = self.hour % 24 # - smj
        # handle hours between noon and midnight
        if self.hour >= 12: # - smj
            am_or_pm = 'PM' # - smj
            # do not substract 12 for noon
            if self.hour != 12: # - smj
                self.hour = self.hour-12 # - smj
        # display 12 as hour unit for midnight
        if self.hour == 00: # - smj
            self.hour = 12 # - smj
        # return string with single digit minutes having a leading zer
        # and without the seconds unit
```

```
        return '%d:%.2d %s' % (self.hour, self.minute, am_or_pm) # - s

    def print_time(self):
        print(str(self))

    def time_to_int(self):
        """Computes the number of seconds since midnight."""
        minutes = self.hour * 60 + self.minute
        seconds = minutes * 60 + self.second
        return seconds

    def is_after(self, other):
        """Returns True if t1 is after t2; false otherwise."""
        return self.time_to_int() > other.time_to_int()

    def __add__(self, other):
        """Adds two Time objects or a Time object and a number.

        other: Time object or number of seconds
        """
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)

    def __radd__(self, other):
        """Adds two Time objects or a Time object and a number."""
        return self.__add__(other)

    def add_time(self, other):
        """Adds two time objects."""
        assert self.is_valid() and other.is_valid()
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)

    def increment(self, seconds):
        """Returns a new Time that is the sum of this time and seconds
        seconds += self.time_to_int()
        return int_to_time(seconds)

    def is_valid(self):
        """Checks whether a Time object satisfies the invariants."""
        if self.hour < 0 or self.minute < 0 or self.second < 0:
            return False
        if self.minute >= 60 or self.second >= 60:
            return False
        return True


def int_to_time(seconds):
    """Makes a new Time object.

    seconds: int seconds since midnight.
    """
    minutes, second = divmod(seconds, 60)
    hour, minute = divmod(minutes, 60)
```

```
    time = Time(hour, minute, second)
```

In [318]:
```
# Test some of the features of Class Time - jdp
def main():     # jdp
    start = Time(9, 45, 00)
    start.print_time()

    end = start.increment(1337)
    end.print_time()

    print('Is end after start?', end=" ")
    print(end.is_after(start))

    # Testing __str__
    print(f'Using __str__: {start} {end}')

    # Testing addition
    start = Time(9, 45)
    duration = Time(1, 35)
    print(start + duration)
    print(start + 1337)
    print(1337 + start)

    print('Example of polymorphism')
    t1 = Time(7, 43)
    t2 = Time(7, 41)
    t3 = Time(7, 37)
    total = sum([t1, t2, t3])
    print(total)

    # A time that is invalid
    t1 = Time(50)
```

In [319]:
```
```

```
9:45 AM
10:07 AM
Is end after start? True
Using __str__: 9:45 AM 10:07 AM
11:20 AM
10:07 AM
10:07 AM
Example of polymorphism
11:01 PM
2:00 AM
```

Your tests

Put your tests in the cell below. These might be assertions, or might be simple print statements

You should have at least three tests

In [320]:
```
def test_time():
    # Test __str__()
    # test problem statement
```

```
    assert Time(17, 30, 0).__str__() == "5:30 PM"
    # noon
    assert Time(12,00,00).__str__() == "12:00 PM"
    # midnight
    assert Time(00,00,00).__str__() == "12:00 AM"
    print('All tests passed!')
```

```
All tests passed!
```

List your changes

List the changes you made in the cell below. This is easy to do if you have signed all your edits.

If you didn't sign, refer to your backup of the original and compare line by line or use a diff function

If you didn't make a backup, download the assignment again and compare the original with your version

In [321]:

```
# Modify this routine - jdp
def __str__(self):
    am_or_pm = 'AM' # - smj
    # keep hours under 24
    if self.hour > 24: # - smj
        # if greater, process days and find remaining hours
        self.hour = self.hour % 24 # - smj
    # handle hours between noon and midnight
    if self.hour >= 12: # - smj
        am_or_pm = 'PM' # - smj
        # do not substract 12 for noon
        if self.hour != 12: # - smj
            self.hour = self.hour-12 # - smj
    # display 12 as hour unit for midnight
    if self.hour == 00: # - smj
        self.hour = 12 # - smj
    # return string with single digit minutes having a leading zero
    # and without the seconds unit
```

Unit Test

In [322]:

```
def test_time():
    # Test __str__()
    print(Time(0, 0, 0))
    print(Time(0, 1, 2))
    print(Time(11, 30, 59))
    print(Time(12, 0, 3))
    print(Time(23, 2, 2))
    print()

    # Test changes to keep time within 24 hours
    print(Time(25, 45, 00))
    print(Time(20, 45, 00) + Time(20, 45, 00))
    print(Time(23, 45, 00) + 72000)
```

```
    print(72000 + Time(23, 45, 00))
    print(Time(25, 45, 00).increment(72000))
    print(int_to_time(180000))
```

```
12:00 AM
12:01 AM
11:30 AM
12:00 PM
11:02 PM

1:45 AM
5:30 PM
7:45 PM
7:45 PM
9:45 PM
2:00 AM
```