Exam 2

Q1

We are creating a list of `(x, y)` coordinate pairs (represented as tuples), based on calculating the `math.sin` function of a given sequence of `x` values. (You can assume that the `math` module has been imported)

For example, given:

In [1]:
```
x = [1, 2, 3, 4, 5, 6]
```

The return value of calling the function `gen_coords(math.sin, x)` should return:

```
[(1, 0.8414709848078965),
 (2, 0.9092974268256817),
 (3, 0.1411200080598672),
 (4, -0.7568024953079282),
 (5, -0.9589242746631385),
 (6, -0.27941549819892586)]
```

Which of the following functions will work correctly?

I.

```
def gen_coords(fn, x):
    return list(zip(x, map(fn, x)))
```

II.

```
def gen_coords(fn, x):
    return [(v, fn(v)) for v in x]
```

III.

```
def gen_coords(fn, x):
    return [(v, map(fn, v)) for v in x]
```

- a. I only
- b. I and II only

- c. I and III only
- d. all of them

Q2

The following collection of dictionaries has to be filtered to only include the elements with a `symbol` field equal to `ABC`, and ordered by the difference between high and low in descending order.

In [2]:

```
data = [
    {
        'symbol': 'ABC',
        'date': '1900-01-01',
        'high': 100,
        'low': 50
    },
    {
        'symbol': 'ABC',
        'date': '1900-01-02',
        'high': 90,
        'low': 60
    },
    {
        'symbol': 'DEF',
        'date': '1900-01-01',
        'high': 100,
        'low': 50
    },
    {
        'symbol': 'ABC',
        'date': '1900-01-03',
        'high': 120,
        'low': 10
    },
```

```
    {
        'date': '1900-01-04',
        'high': 100,
        'low': 90
    }
]
```

For this data set, the result should be:

```
[
    {
        'symbol': 'ABC',
        'date': '1900-01-03',
        'high': 120,
        'low': 10
    },
    {
        'symbol': 'ABC',
        'date': '1900-01-01',
        'high': 100,
        'low': 50
    },
    {
        'symbol': 'ABC',
        'date': '1900-01-02',
        'high': 90,
        'low': 60
    }
]
```

Which of the following functions will return the desired result?

I.

```
def extract(data):
    pred = lambda d: d.get('symbol') == 'ABC'
    filtered = filter(pred, data)
    return list(sorted(filtered, key=lambda d:
d['high'] - d['low'], reverse=True))
```

II.

```
def extract(data):
    filtered = filter(lambda d: d['symbol'] ==
'AAPL', data)
    return list(sorted(filtered, key=lambda d:
d['high'] - d['low'], reverse=True))
```

III.

```
def extract(data):
    filtered = []
    for d in data:
        if d.get('symbol') == 'ABC':
            filtered.append(d)
    sort_func = lambda d: d['high'] - d['low']
    sorted_data = sorted(filtered, key=sort_func,
reverse=True)
    return list(sorted_data)
```

- a. I only
- b. I and II only
- c. I and III only
- d. all of them

Q3

The following function is meant to be used as a decorator that will convert the result of a function to an ISO formatted date string if the function returns a `datetime` object, and leave the result unaffected otherwise.

```
1 from datetime import datetime
2
3 def normalize_datetime(fn):
4     def inner(*args, **kwargs):
5         result = fn(args, kwargs)
6         if isinstance(result, datetime):
7             result = result.isoformat()
8         return fn
9     return result
```

However, there are a few things wrong with this function. Which of the following changes should be made to this function for it to work properly:

I. Line 5 should be changed to: `result = fn(*args, **kwargs)`

II. Line 8 should be changed to: `return result`

III. Line 9 should be changed to: `return inner`

- a. I and II only
- b. II and III only
- c. I and III only
- d. all of them

Q4

What format string should be used to parse the following string into a `datetime` object using the `datetime.strptime()` function:

```
t = '8:30pm on the 3rd of March, 2020'
```

- a. `'%H:%m%p on the %Drd of %M, %Y'`
- b. `'%I:%M%p on the %drd of %B, %Y'`
- c. `'%h:%h%p on the %drd of %b, %Y'`
- d. `'%I:%M%P on the %drd of %B, %Y'`

Q5

You are writing some code to append lines of text to an existing text file.

Which variant of the `open` function should you use:

- a. `open(f, 'r')`
- b. `open(f, 'w')`
- c. `open(f, 'a')`
- d. `open(f, 'rw')`

Q6

A CSV file has rows formatted as follows:

```
'/string-1/- string2- string3-100'
```

Parsing this row should result in the fields `['string-1', ' string2', ' string3', '100']`

What settings should be used when using Python's CSV reader to parse this file correctly?

- a. `csv.reader(file, delimiter='-', quotechar='/', skipinitialspace=False)`
- b. `csv.reader(file, delimiter='-', quotechar='/', skipinitialspace=True)`
- c. `csv.reader(file, delimiter='/', quotechar='-', skipinitialspace=False)`
- d. `csv.reader(file, delimiter='/', quotechar='-', skipinitialspace=True)`

Q7

Which of the following expressions can be used to generate a random `float` in the interval `[15, 25)` (i.e. the random number `x` should satisfy `15 <= x < 25`)

- a. `15 + 25 * random.number()`
- b. `10 + 15 * random.number()`
- c. `40 * random.number() - 15`
- d. `15 + 10 * random.random()`

Q8

For this exercise you will need to first set your random seed to `-1` (`random.seed(-1)`)

Once you have done this, use the `statistics` module to create a normal distribution centered at `3` with a standard deviation of `5`. Determine the median of a `50` element sample from the distribution (use the distribution object's `sample()` method to extract your sample of `50` elements).

The median, rounded to 2 digits after the decimal point is:

- a. `3.0`
- b. `3.83`
- c. `5.0`
- d. `2.99`

Q9

You have this 2-dimensional `Vector` class defined as follows:

In [3]:
```
class Vector:
    def __init__(self, x, y):
        self._x = x
        self._y = y

    @property
    def x(self):
        return self._x

    @property
    def y(self):
        return self._y
```

When you print out an instance of this class, Python's default string representation is shown. You would rather have a more specific representation that should result in the **same output** for **both** `repr()` and `str()` of the instance.

For example, if you have a vector:

In [4]:
```
v = Vector(1, 2)
```

Then you want the following results:

`str(v)` --> `'Vector(1, 2)'`

`repr(v)` --> `'Vector(1, 2)'`

What code can you add to your class to do this:

I.

```
def __str__(self):
    return f'Vector({self.x}, {self.y})'
```

II.

```
def __repr__(self):
    return f'Vector({self.x}, {self.y})'
```

III.

```
def __str__(self):
    return f'Vector({self._x}, {self._y})'
```

IV.

```
def __repr__(self):
    return f'Vector({self._x}, {self._y})'
```

- a. I or II only
- b. III or IV only
- c. II or IV only
- d. any of them will work

Q10

Continuing with the `Vector` class defined in the previous question, we want to implement equality of two instances of `Vector` such that two instances are considered equal if:

- they are both instances of `Vector`
- the `x` components of each vector are equal, and the `y` components of each vector are equal

Once you have implemented this functionality, then:

```
Vector(1, 2) == Vector(1, 2)
```

should evaluate to `True`

Which of the following additions to your class could implement this behavior:

I.

```
def __eq__(self, other):
    return self == other
```

II.

```
def __eq__(self, other):
    return isinstance(other, Vector)
```

III.

```
def __eq__(self, other):
    if (
        isinstance(other, Vector)
        and self.x == other.x
        and self.y == other.y
    ):
        return True
    return False
```

- a. I only
- b. II only
- c. III only
- d. none of them