SSW 810A Final Exam

Due Dec 29, 2020 at 11:59pm **Points** 100 **Questions** 24
Available Dec 28, 2020 at 8am - Dec 29, 2020 at 11:59pm 1 day
Time Limit 150 Minutes

This quiz was locked Dec 29, 2020 at 11:59pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	150 minutes	100 out of 100

Correct answers are hidden.

Score for this quiz: **100** out of 100
Submitted Dec 6, 2020 at 1:34pm
This attempt took 150 minutes.

Question 1 — 4 / 4 pts

Compare and contrast Relational databases and NoSQL databases such as MongoDB.

Your Answer:

The two main types of modern databases to choose from are relational and non-relational, also known as SQL and NoSQL.

- SQL databases are known as relational databases, and have a table-based data structure, with a strict, predefined schema required.
- NoSQL databases or non-relational databases can be document based, graph databases, key-value pairs, or wide-column stores.
- NoSQL databases don't require any predefined schema, allowing you to work more freely with **unstructured data.**
- Relational databases are vertically scalable, but usually more expensive, whereas the horizontal scaling nature of NoSQL databases is more cost-efficient.

- NoSQL databases feature dynamic schema and allow you to use what's known as **unstructured data.** This means you can build your application without having to first define the schema. In a relational database, you are required to define your schema before adding data to the database. Not needing a predefined schema makes NoSQL databases much easier to update as data and requirements change.
- Relational databases are table-based. NoSQL databases can be document based, graph databases, key-pairs, or wide-column stores. Relational databases were built during a time that data was mostly structured and clearly defined by their relationships.
- NoSQL databases tend to be more a part of the open-source community. Relational databases are typically closed source with licensing fees baked into the use of their software.

Question 2

4 / 4 pts

Explain why try blocks in try/except statements should be kept small. What can go wrong in big try blocks?

Your Answer:

Try blocks and excepts need to be kept small due to various reasons. The primary reason behind the same is if we perform multiple operations inside a single try block then it would be difficult for us to narrow down to the actual cause behind the error and we will have to go line by line in order to find the actual cause. If we have multiple try blocks for every operation like reading a file, passing a value to a function, and performing some kind of operation then it will become easier for us to find which block exactly failed and a high level context in which the call was made.

Try blocks should be kept small with few statements to help to isolate the cause of the exception.

Question 3

4 / 4 pts

Describe how ***branches*** are used when using Git or GitHub.

Your Answer:

- The way Git and GitHub manage the timelines of versions of a project especially when more than one person is working on the project and making changes is by using branches.
- A branch is essentially a unique set of code changes with a unique name. Each repository can have one or more branches. The main branch is the one where all changes eventually get merged back into, is called master. This is the official working version of your project and one of you when you visit the project repository at github.com/yourname/projectname.
- Branches allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.
- You always create a branch from an existing branch. Typically, you might create a new branch from the default branch of your repository. You can then work on this new branch in isolation from changes that other people are making to the repository. A branch you create to build a feature is commonly referred to as a feature branch or topic branch.

A ***branch*** isolates changes in a ***specific context***

Create new features, do bug fixes, experiment, …

e.g. Master branch may represent the current product

Dev branch may represent change for a group of features

Question 4

4 / 4 pts

Describe Test Driven Development (TDD). How does TDD help programmers?

Your Answer:

Test-driven development (TDD) is a software development process relying on software requirements being converted to test cases before the software is fully developed, and tracking all software development by repeatedly testing the software against all test cases.

When a test fails, you have made progress because you know that you need to resolve the problem. TDD ensures that your system actually meets the requirements defined for it. It helps to build your confidence in your system. In TDD more focus is on production code that verifies whether testing will work properly.

Write tests before writing code

Question 5

4 / 4 pts

Why is it important to test boundary conditions? Include three boundary conditions for a function that calculates the absolute value of a number.

Your Answer:

It is very important to test boundary conditions because it is the process of testing between partitions of the input values. So these extreme ends like Start-End, Lower-Upper, Minimum-Maximum, Just Inside-Just Outside values are called boundary values and are very important to take into consideration when programming.

- It is used to find the errors at the boundaries of the input domain rather than finding those errors in the center of the input.

The absolute value of a number is the determinant of the number eg. |-1| = |1| = 1

if the function is called find_abs(a)

the test cases for the values of a must be as such

1) checking if a is a number as all: if a is not a number we should raise a ValueError.

with self.assertRaises(ValueError):find_abs("hello")

2) checking for negative numbers: if a negative number is passed then it should return it's positive counterpart.

self.assertEqual(find_abs(-1),1)

3) checking for a positive numbers: if a positive number is passed it should return the same number.

self.assertEqual(find_abs(1),1)

Question 6

4 / 4 pts

Explain how ***pull requests*** are helpful when working on open source projects.

Your Answer:

Pull requests let you tell others about changes you have pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with the collaborator and follow-up commits before your changes are merged into the base branch. Since it is an open-source project, pull requests can let all collaborators know what changes you would want to push to the code. This way code can be easily reviewed and changes can be implemented.

Pull requests allow everyone on the team to review changes before the changes are merged into the Master Branch

You are asking the team to pull your changes into the Master Branch

GitHub provides a collaborative environment to allow teams to work together in the same room or geographically dispersed.

Question 7

4 / 4 pts

Describe a scenario where a Python decorator is appropriate and helpful. You ***do not*** need to write the code, just describe a situation where a decorator is helpful.

Your Answer:

- Decorators are a very powerful and useful tool in Python since it allows programmers to modify the behavior of function class. Decorators allow us to wrap another function in order to extend the behavior of the wrapped function, without permanently modifying it.
- In decorators, functions are taken as the argument into another function and then called inside the wrapper function.
- Especially with large applications, we often need to specifically measure what's going on, and record metrics that quantify different activities. By encapsulating such noteworthy events in their own function or method, a decorator can handle this requirement very readily and easily

Time a function call

Track how many times a function is called

Memoize a function to use previous results

Mock a function call while testing

Web frameworks, e.g. Flask

Question 8

5 / 5 pts

Write a function *dup(my_list) t*hat uses a list comprehension to return a copy of my_list, the value passed to *dup()*.

You **MAY** use your favorite Python IDE to solve this problem
You do **NOT** need to write test cases.

Your Answer:

```
def dup(my_list: List) -> List:
    output = [item for item in my_list]
```

return output

```
def dup(my_list: List[Any]) -> List[Any]:
    return [val for val in my_list]
```

Question 9

5 / 5 pts

Write a function remove_spaces(str) that takes a string as a parameter and returns a copy of str after removing all spaces. E.g.

```
remove_spaces("hello world") == "helloworld"
remove_spaces("this has several spaces") == "thishasseveralspaces"
```

You **MAY** use your favorite Python IDE to solve this problem. You do NOT need to write test cases.

Your Answer:

def remove_spaces(input: str) -> str:
"""
remove space from string
"""
result: str = input.replace(" ", "")
return result

```
def remove_spaces(s: str) -> str:
    return "".join([c for c in s if c != ' '])

or
```

```
def remove_spaces(s: str) -> str:
    """takes a string as a parameter and returns a copy of str aft
er removing all spaces"""
    return s.replace(" ", "")
```

Question 10

5 / 5 pts

Write a **generator**

```
my_range(start, limit, increment=1)
```

That takes integers ***start***, and ***limit*** along with an optional integer ***increment*** that mimics the behavior of Python's range() function. Your generator **MAY NOT** use the Python range() function.

E.g.

```
list(my_range(-1, 3)) == [-1, 0, 1, 2]
list(my_range(3, 7, 2)) == [3, 5]
```

You **MAY** use your favorite Python IDE to solve this problem
You do **NOT** need to write test cases.

Your Answer:

```
def my_range(start: int, limit: int, increment: int = 1) -> int:
    """
    my range generates numbers in between start and limit taking incre
ment into consideration
    """
    i = start
    while i < limit:
        yield i
        i += increment
```

```
def my_range(start: int, end: int, increment: int=1) -> Iterator[i
nt]:
    """ Simulate range(start, end, increment=1) """
    if increment >= 0:
        while start < end:
            yield start
            start += increment
    else:
        while start > end:
            yield start
            start += increment

class MyRangeTest(unittest.TestCase):
    def test(self):
        self.assertEqual(list(my_range(0, 3)), [0, 1, 2])
        self.assertEqual(list(my_range(0, 3, 2)), [0, 2])
        self.assertEqual(list(my_range(4, 3)), [])
        self.assertEqual(list(my_range(3, -1, -1)), [3, 2, 1, 0])
        self.assertEqual(list(my_range(0, 3, -2)), [])
```

Question 11

5 / 5 pts

Write a function sum_first_n_squares(n) that uses a list comprehension to calculate the sum of the squares from 1 to n, inclusive. E.g. sum_first_n_squares(3) == 1^2 + 2^2 + 3^2 == 14

You **MAY** use your favorite Python IDE to solve this problem
You do **NOT** need to write test cases.

Your Answer:

```
def sum_first_n_squares(num: int) -> int:
    """
    Sum of squares using list comprehension
    """
    if num < 1:
        raise ValueError("You have entered invalid value")
    return sum([int(num*num) for num in range(num+1)])
```

```
def sum_first_n_squares(n: int) -> int
    """ Calculate the sum of the squares from 1 to n """
    return sum([i * i for i in range(1, n + 1)])
```

Question 12

5 / 5 pts

Write a function offsets(seq) that takes a sequence, e.g. a list or string, as a parameter and returns a dictionary where the key is each unique element in the sequence and the value is a list of offsets within the sequence, e.g.

```
offsets("mississippi") ==

{'i': [1, 4, 7, 10], 'm': [0], 'p': [8, 9], 's': [2, 3, 5, 6]}

where 'i' occurs at offset 1, 4, 7, 10, 'm' occurs only at offset 0, etc.

You MAY use your favorite Python IDE to solve this problem.
You do NOT need to write test cases.
```

Your Answer:

```
def offsets(seq) -> Dict:
    result = DefaultDict(list)
    for offset, item in enumerate(seq):
        result[item].append(offset)
    return result
```

```
def offsets(seq: Sequence[Any]):
    result: DefaultDict[str, List[int]] = defaultdict(list)

    for offset, item in enumerate(seq):
        result[item].append(offset)

     return result
```

Question 13

5 / 5 pts

Write a **generator** up_down(n) that yields values from 0 up to n and then back down to 0, e.g.

```
list(up_down(3)) returns
[0, 1, 2, 3, 2, 1, 0]

Be sure to include type hints.
```

You **MAY** use your favorite Python IDE to solve this problem
You do **NOT** need to write test cases.

Your Answer:

```
def up_down(n: int) -> Iterator[int]:
    """Count up to n and go back to zero

    Args:
        n (int): [description]

    Yields:
        Iterator[int]: [description]
    """
    yield from range(n)
    yield from range(n, -1, -1)
```

```
def up_down(n: int) -> Iterator[int]:
    yield from range(n)
    yield from range(n, -1, -1)
```

Question 14

4 / 4 pts

It's important to follow Python coding guidelines because:

(choose all that apply)

- [x] Coding guidelines improve readability and maintainability
- [ ] You don't want to look like a Java programmer
- [ ] It's not important: do whatever you like
- [ ] Because the reader should have as much trouble reading the code as I had writing it

Question 15

4 / 4 pts

Select all of the reasons for refactoring:

- [ ] refactoring creates "technical debt"
- [x] refactoring improves the design
- [x] refactoring makes code easier to understand and change
- [ ] refactoring obscures bugs

Question 16

4 / 4 pts

Python determines the type of a variable

- ( ) at Superbowl halftime
- (•) dynamically at run time
- ( ) statically at compile time
- ( ) statically while reading the .py file

Question 17

4 / 4 pts

Match the task on the left with the ***most appropriate/optimal*** Python data structure on the right.

swapping values	tuple
queue/stack	list
intersection	set
map one value to another	dictionary

Question 18

4 / 4 pts

Match the variable name to the context using the Python PEP-8 naming guidelines

lower_case_with_undersc	variable, function, or m
CapitolWords	class names and/or ex
ALL_CAPS_WITH_UNDER	global constants

Question 19

4 / 4 pts

All of Python's built in methods for lists are also available for tuples.

- True
- False

Question 20

4 / 4 pts

Choose the optimal Python container to solve a problem where you are given a sequence of names and you must identify the distinct names in the sequence.

- list
- defaultdict(list)
- dict

  Set would be perfect, but since set wasn't offered as an option, one could add each element in the sequence to the dict as a key (with no value) and then just return dict.keys() as the set of unique names.

- tuple

Question 21

4 / 4 pts

What is the most appropriate Python container to use to find the distinct elements in a sequence.

- Tuple

- (•) Set
- ( ) Dict
- ( ) defaultdict
- ( ) List

Question 22

10 / 10 pts

The United States measures weights in pounds and ounces, where 1 pound == 16 ounces.

Write a class Weight to support the following operations:

```
print(Weight(1, 0)) # prints 1.0 pounds 0.0 ounces
print(Weight(1, 16))  # prints 2.0 pounds 0.0 ounces
print(Weight(0, 37)) # prints 2 pounds 5 ounce

w1 = Weight(1, 12)
w2 = Weight(2, 11)
print(w1 + w2) # prints 4 pounds 7 ounces
```

Hints:

- define __str__(self) and __add__(self, other) methods
- 13 % 12 == 1(mod)
- 25 // 12 == 2 (floor division)

You **DO NOT** need to write automated test cases

You **MAY** use your favorite Python IDE to solve this problem.

Your Answer:

```
class Weight:
    def __init__(self, pound: int, ounce: int) -> None:
        self.pound: int = pound + ounce//16
        self.ounce: int = ounce % 16
        self.label: str = f"{self.pound} pounds and {self.ounce} ounce"
```

```
    def __str__(self) -> str:
        """ return a String to display Weights """
        return f"{self.label}"

    def __add__(self, other: "Weight") -> "Weight":
        """ Add two Weights using simplest approach.
            Calculate new numerator and denominator and return new Wei
ght
        """
        pound: int = (self.pound + other.pound) + \
            (self.ounce + other.ounce)//16
        ounce: int = (self.ounce + other.ounce) % 16

        result: Weight = Weight(pound, ounce)
        return result
```

```
class Weight:
    """ US weight calculations """
    def __init__(self, pounds:float, ounces: float):
        self._pounds: float = pounds + (ounces // 16)
        self._ounces: float = ounces % 16

    def __add__(self, other: "Weight"):
        return Weight(self._pounds + other._pounds, other._ounces
 + self._ounces)

    def __str__(self):
        return f"{self._pounds:.1f} pounds {self._ounces:.1f} ounc
es"
```

Unanswered

Question 23

0 / 0 pts

If needed, use this space to explain any of your answers for any of the items used in this quiz. Please note the item number with your comment.

Your Answer:

Question 24

0 / 0 pts

“I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination. I further pledge that I have not copied any material from a book, article, the Internet or any other source except where I have expressly cited the source.”

- (●) True
- ( ) False

Quiz Score: **100** out of 100