Quiz 02

Due Sep 20 at 10pm **Points** 10 **Questions** 5 **Time Limit** None

Instructions

Answer the following questions in your own words. Do NOT simply cut and paste the information from the slides. You will receive a score of 0 if you copy the prose from the slides.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	31 minutes	0 out of 10 *

* Some questions not yet graded

Score for this quiz: **0** out of 10 *
Submitted Sep 20 at 8:17pm
This attempt took 31 minutes.

Question 1

Not yet graded / 2 pts

How can Python programmers avoid adding instance attributes caused by typing mistakes?

Your Answer:

We can use **"__slots__"** to specify the attributes of the class.

Use the __slots__ attribute to explicitly specify the attributes for the class.

Question 2

Not yet graded / 2 pts

I want to define a class 'Quiz' that is initialized with a single parameter x. Python didn't complain when I defined the class but I get an error "**TypeError: object() takes no parameters"** when I try to create an instance of class Quiz as show below.

```
class Quiz:
    def init(self, score: int):
        self.grade: int = score

>>> q = Quiz(10)

TypeError: Quiz() takes no arguments
```

Fix the code so I can create and initialize instances of class Quiz properly.

Your Answer:

we need to add "**__**" before and just after init in order to make the program work.

Rectified Code:

```
def __init__(self, score: int):
        self.grade: int = score
```

Quiz.init(score: int) should be named Quiz.__init__(self, score:int)

Question 3

Not yet graded / 2 pts

Explain how encapsulation can help to improve your code. Why should you use encapsulation? What can go wrong if you don't?

Your Answer:

Encapsulation can help us to improve our code in the following ways:

- Encapsulation helps in enhancing the consistency and predictability.
- Encapsulation makes the code easier to understand, maintain, and reuse.
- It makes functions and objects independent of each other.
- Not using encapsulation restricts our ability to independently changing the modules of the code.
- Data security could be limited if encapsulation is not implemented.

Question 4

Not yet graded / 2 pts

Describe a situation where you might raise an exception. Write the code to raise a ValueError exception to warn users about an invalid value, x, which is negative, but should be >= 0

Your Answer:

Let's take into consideration the get_number function which validates the input of a fraction is of type float. The input of a fraction(Numerator or Denominator) can never contain any sort of alphabets.
Example code:

```
def get_number(prompt: str) -> float:
    """
    In this function, we take the input of each part of the Fraction (Numer
ator & Denominator), validate the same and return the input received fr
om the User.
    """
    loop: bool = True
    while loop:
        try:
            input1: str = input(prompt)
            return float(input1)

        except ValueError as e:
            print("Please enter only numeric Value, Try again!")
            continue
```

```
if x < 0:
    raise ValueError(f"x has value {x} but must be >= 0")
```

Question 5

Not yet graded / 2 pts

What is the output of the following code:

```
def raise_exception():
    raise ValueError
    print("leaving raise_exception()")

def inner():
    raise_exception()
    print("leaving inner()")

def outer():
    inner()
    print("leaving outer()")

def way_out():
    try:
        outer()
    except ValueError:
        print("way_out(): caught a ValueError")
    print("leaving way_out()")

way_out()
```

Your Answer:

```
way_out(): caught a ValueError
```

```
leaving way_out()


Flow of executation:
```

```
way_out() -> outer() -> inner() -> raise_exception() -> raise ValueError
->
```

```
print("way_out(): caught a ValueError") -> print("leaving way_out()")
```

way_out(): caught a ValueError leaving way_out()

way_out() calls outer().
outer() calls inner().
inner() calls raise_exception().
raise_exception() raises a ValueError exception that is not caught in raise_exception() so Python checks in inner(). Inner() does not catch the exception so Python pops the stack and returns to outer(). Outer() does not catch the exception so control returns to way_out(). way_out() catches the exception, prints the "caught a ValueError" message then executes the next line which prints the "leaving way_out()" message.

Quiz Score: **0** out of 10