Results

Shang Xiao — 1st Attempt

100%

10
Out of 10 points

253:19:24
Time for this attempt

This assessment has unlimited attempts. Take Now

Attempt History

Results	Points	Score	(Highest score is kept)
Attempt 1	10 of 10	100%	(Highest score)

Your Answers:

1 3 / 3 points

What are the three methods of dealing with incorrect information?

Two general ideas are asking permission and asking forgiveness. The key difference between asking these two general methods is at what stage do we check for potential misbehavours of our codes.

Asking for permissions usually check code errors before we execute them by using conditional statements to generate an error or by using loops to repeat a code block until user input is correct. This approach is, quote from Professor Maira, "good to problems that are predictable and common."

Asking forgiveness is by trying to execute the entire code block altogether using the *try* and *except* statements and then hoping that we can handle all potential errors that could have been raised at the end altogether. From my personal perspective, this approach is more common in real life and is explained in the Python official documentation in the exception/error handling module.

There are two ways mentioned in our couse module under asking for permissions, one is by raising errors, for example:

```
def get_slices(pies, folks):
    if not isinstance(folks, int) or folks <= 0:
        raise ValueError("folks must be a positive integer")
    if not isinstance(pies, int):
        raise TypeError("pies must be an integer value")
    if pies < 0:
        raise ValueError("pies must be non-negative integer")
    # multiply by 8 slices per pie and divide
    slices = pies * 8 // folks
    return slices
```

Another one method for asking permissions is with repetition, for example, we keep promopting the user to enter/correct their input until they give us the correct type and value we need:

```
def main():
    while pizzas < 0:
        pizzas_text = input("How many pizzas did you order? ")
        if pizzas_text.isnumeric():
            pizzas = int(pizzas_text)

    people = -1
    while people <= 0:
        people_text = input("How many people are there? ")
        if people_text.isnumeric():
            people = int(people_text)

    slices = get_slices(pizzas, people)
    print(pizzas, "pizzas split", people, "ways is", slices, "slices each")


main()
```

Lastly, asking forgiveness, using the *try* and *except* statements:

```
def main():
    try:
        pizzas = int(input("How many pizzas did you order? "))
        people = int(input("How many people are there? "))
        slices = get_slices(pizzas, people)
        print(pizzas, "pizzas split", people,
              "ways is", slices, "slices each")

    except TypeError as ex:
        print("Invalid type:", type(ex), ex)
    except ValueError as ex:
        print("Invalid value:", type(ex), ex)


main()
```

One thing I also learned is that sometimes is good practice to use more than one of these three methods we mentioned above to write defensive codes, for example, in Professor Maria's example, she implemented error handling technique in both get_slices() funtion and main() function, in this case it is not a repetition but a good habit to have, since anyone can access both functions, it is always better to keep these error handling codes within both functions.

Correct

2 2 / 2 points

What are 2 of our current assumptions about users?

Our current assumptions about users are that they make mistakes and are not smart anymore, also, they don't know what the computer wants and they have zero knowledge about data types and value. We have to be extremely specific about the information we require from them when prompting them, at the current stage.

Correct

3 2 / 2 points

What are the two types of exceptions discussed in class?

TypeError and ValueError, for example:

```
def readNumbers():
    try:
        num = input("Enter a number")
        number = int(num)
        if (num == "4"):
            print("Number 4")
        print("A" + number)
    except TypeError as ex:
        print("Invalid type", type(ex), ex)
    except ValueError as ex:
        print("Invalid value", type(ex), ex)
```

When we enter 2, raise TypeError: Invalid type <class 'TypeError'> can only concatenate str (not "int") to str'

When we enter two, raise ValueError: Invalid value <class 'ValueError'> invalid literal for int() with base 10: 'two'

Correct

4 2 / 2 points

What are two problems that can happen when you try to read information from a file?

There are couple of errors that could be raised while we try to read or open a file using Python.

FileNotFoundError is raised when we try to open a file for reading that does not exist, this can be caused if the file name we typed in Python is inconsistent with the actual file name stored locally.

PermissionError is raised when we try to open a file for reading that the user or program does not have adequate permissions for reading or writing, this is because files can be protected using features such as 'read-only', and if the user has less control power than the administrator of the computer, this can also be the reason that why that user cannot access the file.

OSError is raised when we try to open a file for reading if the file disapperas during this process of the operation of opening a file for reading.

Correct

5 1 / 1 point

What is the line of code that will get the file name that is put in at the command line to be read in?

input = open("xxx.txt')

where 'xxx' refers to the actual file name.

Correct