1DB18IS098 Vivek V. Pai

Python - 2nd assignment.

Group-4

① How Exceptions can be raised using keyword raise() with example.

→ we can raise your own exceptions that you anticipated but we can also raise your, Raising an exception is a way saying. "stop running the code in this function & move the program execution to the except statment"

Exceptions are raised with raise statment In code a raise statment consists of the following:

- The raise keyword.
- A call to the Exception() function.
- A string with a helpful error message passed at the exception() function.

Eg:

```
def forPrint (symbol, width, height):
    if (len(symbol) != 1):
        raise Exception ('symbol must be a
            single character string ")
    if width <= 2:
        raise Exception ('width must be greater than 2')
    if height <= 2
        raise exception ('height must be greater than 2")
    print (symbol * width)
    for (i in range (height - 2):
        print (symbol + (" "* (width - 2)) + symbol
    print (symbol * width).
```

② what are assertions. How to disable assertions explain with example.

→ An assertion is a sanity check to make sure your code isn't doing something obviously wrong. These sanity checks are performed by assert statements. If the sanity checks fails, then an AssertionError exception is raised.

Disabling Assertion.

- Assertion can be disabled by passing the -O option when running Python. This is good for when you have finished writing and testing your program & don't want it to be slowed down by performing sanity checks most of the time. assert statments do not cause a noticeble speed difference.
- Assertions are for development, not the financial product. By the time you hand off your program to someone else to run, it should be free of bugs & not require the sanity checks.

Eg:-

```
def kelvin to Faranheit (Temperature):
    assert (Temperature >= 0)
    return ((Temperature - 273) * 18) + 32
print kelvin to Faranheit (273) → 32.0
print kelvin to Fernhet (-5) → error.
```

(3) what is logging. Explain with example;

Logging is a great way to understand what's happening in your program and in what order its happening. Pythons logging module makes it easy to create a record of custom messages that you write. These log message will describle when the program execution has reached the logging.

③

function call & list any variable you have specified at that point in time.

On the other hand, a missing indicates a part of the code was skipped & never executed.

Eg:- import logging.

logging basic config (level = logging.DEBUG, format = '%(asctime)s - %(levelname)s - %(message)s')

logging debug ('start a program')

def factorial (n):

logging.debug ('start of factorial (%s%%)' %(n))

total = 1

for i in range (n+1):

total *= i

logging debug ('i is '+ str(i) +', total is '+ str(total))

logging.debug ('End of factorial (%s%%)' %(n))

return total

print (factorial (5))

logging.debug ('End of program')

④ what are classes & objects. Explain with example program.

python is an OOP language. An object is simply a collection of data & methods that act on those data similarly. a class is a blueprint for that object.

class.

class definitions begin with a class keyword.

class MyNewClass:

"This is a new class"

pass

A class creates a new local namespace all its attributes are defined. Attributes may be data or functions.

Eg. class Person:
"This is a person class"
age = 10.
def greet (self):
print ("Hello")

print (Person.age) → 10

Objects
It can be used to create new object instances of that class. The procedure to create an objects is similar to a function call.
harry = Person ().

Eg: class Peron:
"This is a perso class"
age = 10.
def greek (self):
print ("Hello").
harry = Person() → Crease a new object of person class.
harry. greet() → "Hello"

⑤ What are IDLE's debegger. Explain all modules of IDLE's debegger.

→ • The debegger is a feature of IDLE that allows you to execute your program one line at a time.

The debugger will run a single line of code and then wait for you to tell it to continue.

- "By running your program" under the debeger" like this.

The program will stay paused until you press one of the five buttons in the Debug control window: Go, step, over, out or Exit.

① Go: Clicking the Go button will cause the program to execute normally until it terminates or reaches a breakpoint.

② Step: Cicking the step button will cause the debegger to execute the next line of code and then pause again

③ Over: clicking the over button will execute the next line of code, similar to the step button, however, if the next line of code is a function call, the over button will "step over" the code in the function.

④. Out: Clicking the out button will cause the debugger to execute lines of code at full speed until it returns from the current function.

⑤ Quit: If you want to stop debugging entirely and not bother to continue executing the rest of the program. click the Quit button.