Assignment Date	18 November 2022
Student Name	G.Karthika
Student Roll Number	815819104011
Maximum Marks	

Exceptions handling

Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).

When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

For example, let us consider a program where we have a function `A` that calls function `B`, which in turn calls function `C`. If an exception occurs in function `C` but is not handled in `C`, the exception passes to `B` and then to `A`.

If never handled, an error message is displayed and our program comes to a sudden unexpected halt.

Catching Exceptions in Python

In Python, exceptions can be handled using a `try` statement.

The critical operation which can raise an exception is placed inside the `try` clause. The code that handles the exceptions is written in the `except` clause.

We can thus choose what operations to perform once we have caught the exception. Here is a simple example.

```
# import module sys to get the type of exception
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!", sys.exc_info()[0], "occurred.")
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

Run Code

Output

```
The entry is a
Oops! <class 'ValueError'> occurred.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError'> occured.
Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

```
# import module sys to get the type of exception
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
```

```
        r = 1/int(entry)
        break
    except Exception as e:
        print("Oops!", e.__class__, "occurred.")
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

```
try:
    # do something
    pass

except ValueError:
    # handle ValueError exception
    pass

except (TypeError, ZeroDivisionError):
    # handle multiple exceptions
    # TypeError and ZeroDivisionError
    pass

except:
    # handle all other exceptions
    pass
```

```
>>> raise KeyboardInterrupt
Traceback (most recent call last):
...
KeyboardInterrupt

>>> raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument

>>> try:
...     a = int(input("Enter a positive integer: "))
...     if a <= 0:
...         raise ValueError("That is not a positive number!")
... except ValueError as ve:
```

```
...     print(ve)
...
Enter a positive integer: -2
That is not a positive number!
```