Dr. Vishwanath Karad

MIT WORLD PEACE UNIVERSITY | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

School of Computer Engineering and Technology

Lab Assignment-03

Write a python program that accepts the length of three sides of a triangle as inputs. The program should indicate whether or not the triangle is a right-angled triangle using function with exception handling.

Python Function

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- Function in Python is defined by the "def " statement followed by the function name and parentheses (())
- Benefits of function
  - Code re-usability
  - Improves Readability
  - Avoid redundancy

Syntax for Function

- Syntax:
  - def name(arguments):
    Statement
    return value

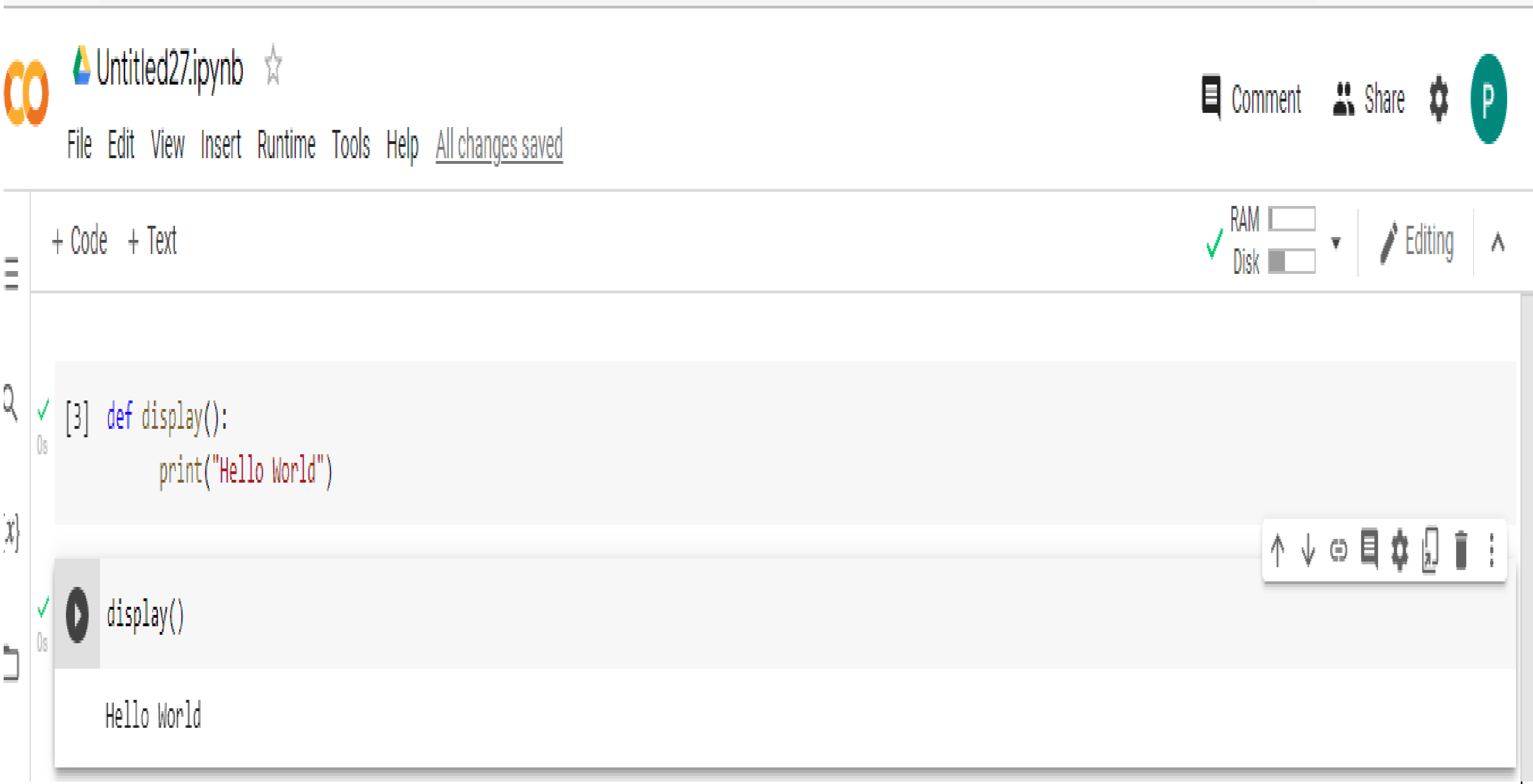
Eg:

```
def display():
        print("Hello World")
```

- Calling defined function

Eg:

```
display()
```



```
[3] def display():
        print("Hello World")
```

```
display()
```

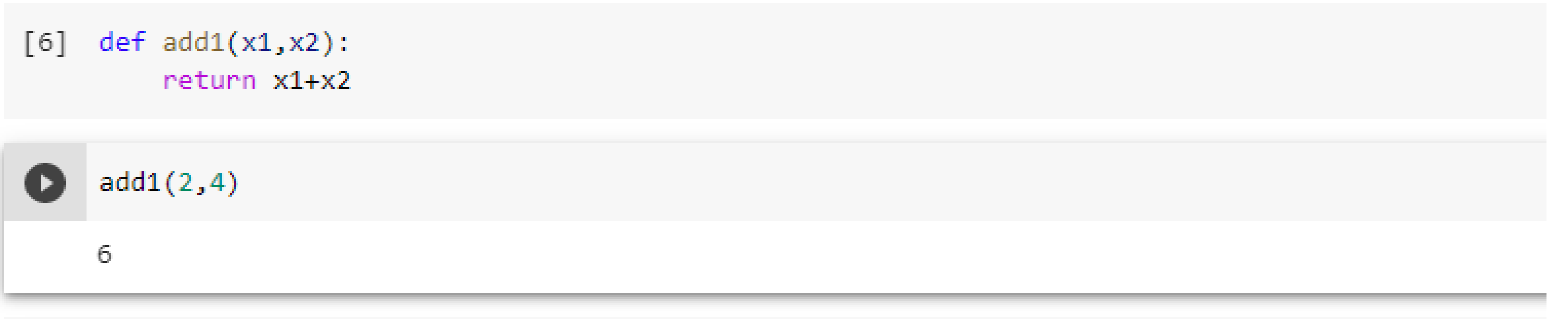
```
Hello World
```

Defining function with return statement in Python:

- def add1(x1,x2):
  return x1+x2

Calling function with a return statement:

add1(10,20)



```
[6] def add1(x1,x2):
        return x1+x2
```

```
add1(2,4)
```

```
6
```

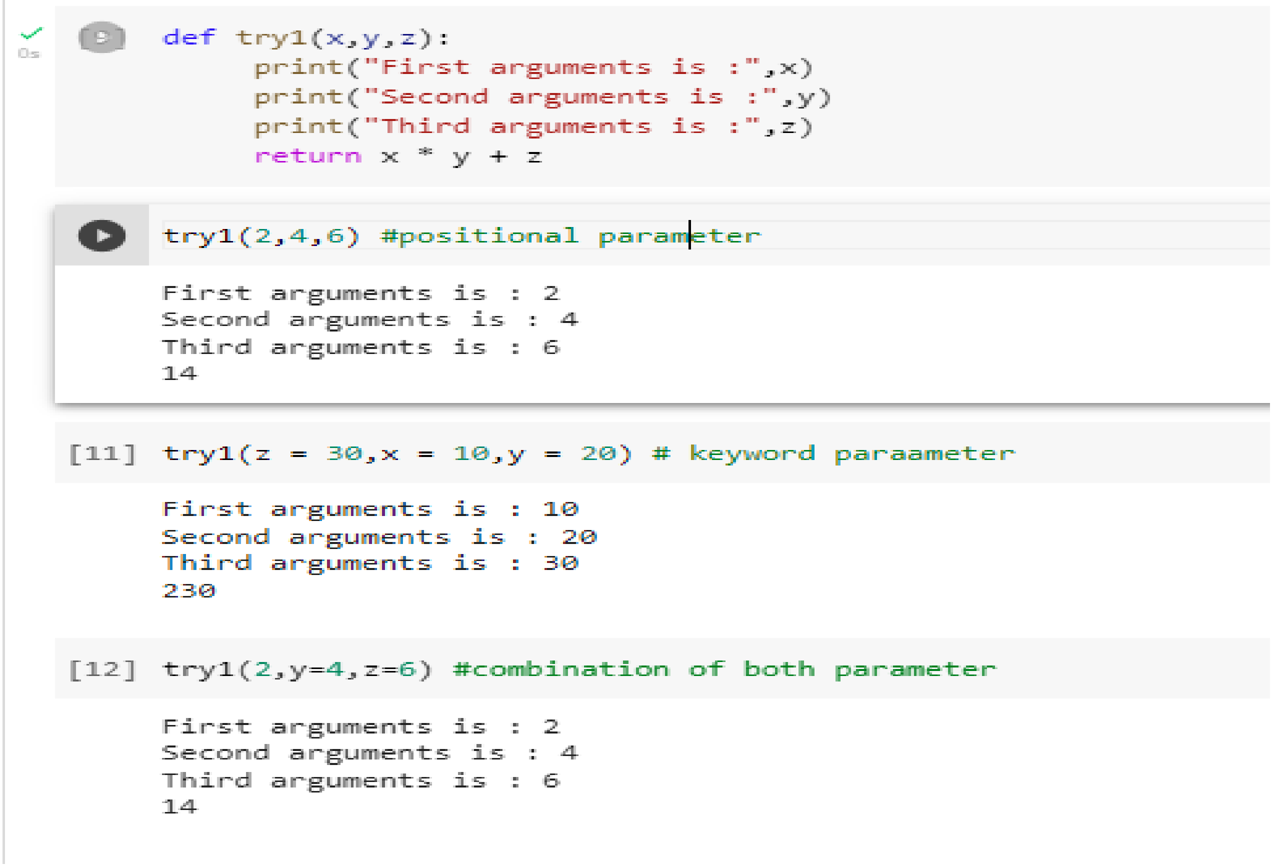
Types of Arguments

- There are two types of argument used when calling a function.

1.Positional Arguments

2.Keyword Arguments

- keyword arguments must follow positional arguments.



```
def try1(x,y,z):
     print("First arguments is :",x)
     print("Second arguments is :",y)
     print("Third arguments is :",z)
     return x * y + z
```

```
try1(2,4,6) #positional parameter
```

```
First arguments is : 2
Second arguments is : 4
Third arguments is : 6
14
```

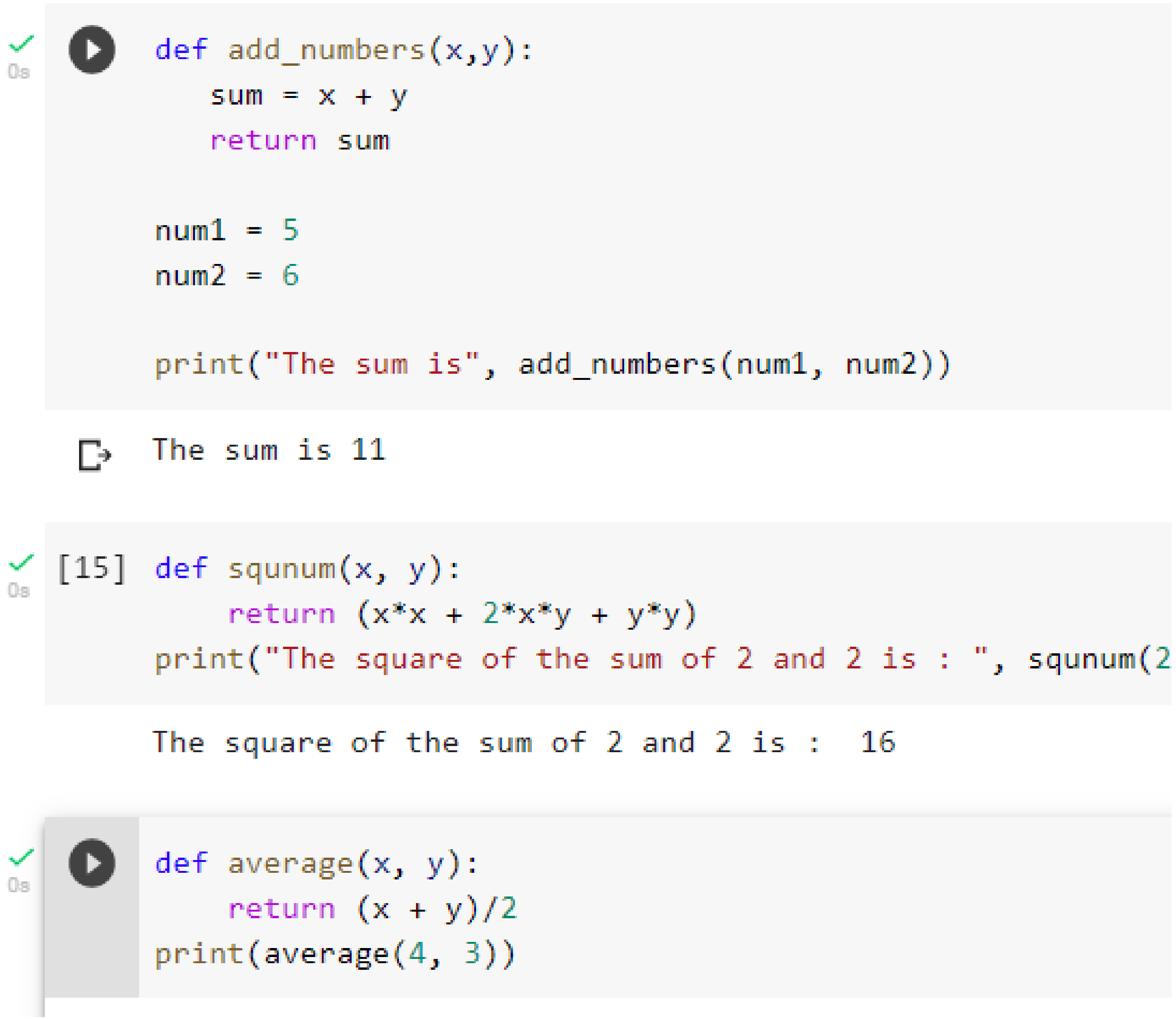
```
[11] try1(z = 30,x = 10,y = 20) # keyword paraameter
```

```
First arguments is : 10
Second arguments is : 20
Third arguments is : 30
230
```

```
[12] try1(2,y=4,z=6) #combination of both parameter
```

```
First arguments is : 2
Second arguments is : 4
Third arguments is : 6
14
```

user-defined function



```
def add_numbers(x,y):
   sum = x + y
   return sum

num1 = 5
num2 = 6

print("The sum is", add_numbers(num1, num2))
```

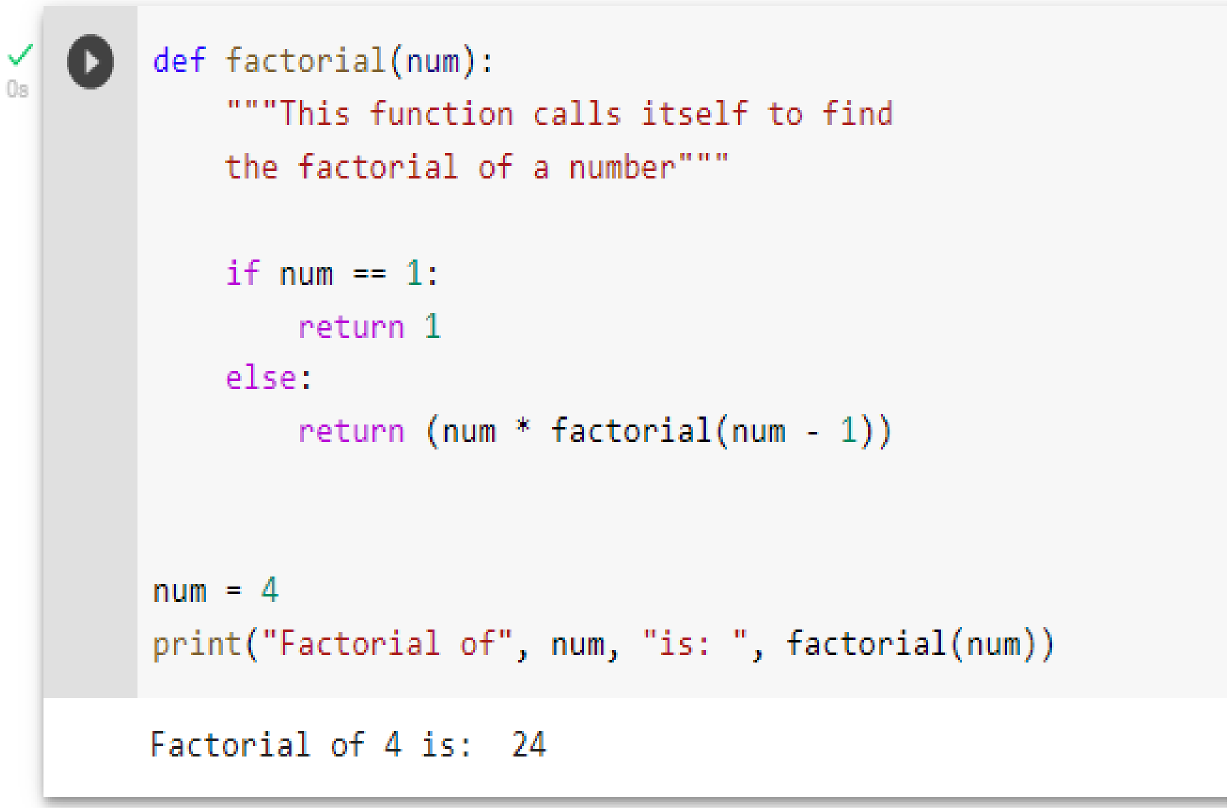
```
The sum is 11
```

```
[15] def squnum(x, y):
         return (x*x + 2*x*y + y*y)
     print("The square of the sum of 2 and 2 is : ", squnum(2
```

```
The square of the sum of 2 and 2 is :  16
```

```
def average(x, y):
     return (x + y)/2
print(average(4, 3))
```

Recursive function

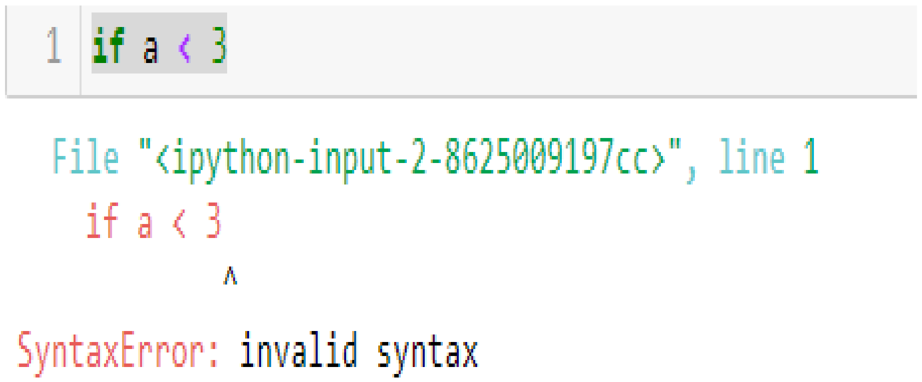


```python
def factorial(num):
    """This function calls itself to find
    the factorial of a number"""

    if num == 1:
        return 1
    else:
        return (num * factorial(num - 1))


num = 4
print("Factorial of", num, "is: ", factorial(num))
```

```
Factorial of 4 is:  24
```

Error and Exceptions

- When writing a program, we, more often than not, will encounter errors.
- Error caused by not following the proper structure (syntax) of the language is called syntax error or parsing error.

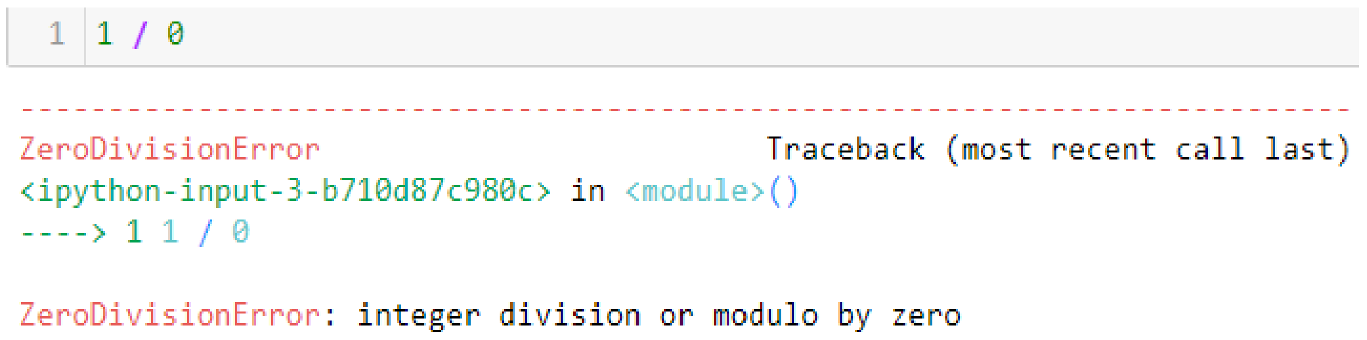

```
if a < 3
```

```
  File "<ipython-input-2-8625009197cc>", line 1
    if a < 3
            ^
SyntaxError: invalid syntax
```

Error and Exceptions

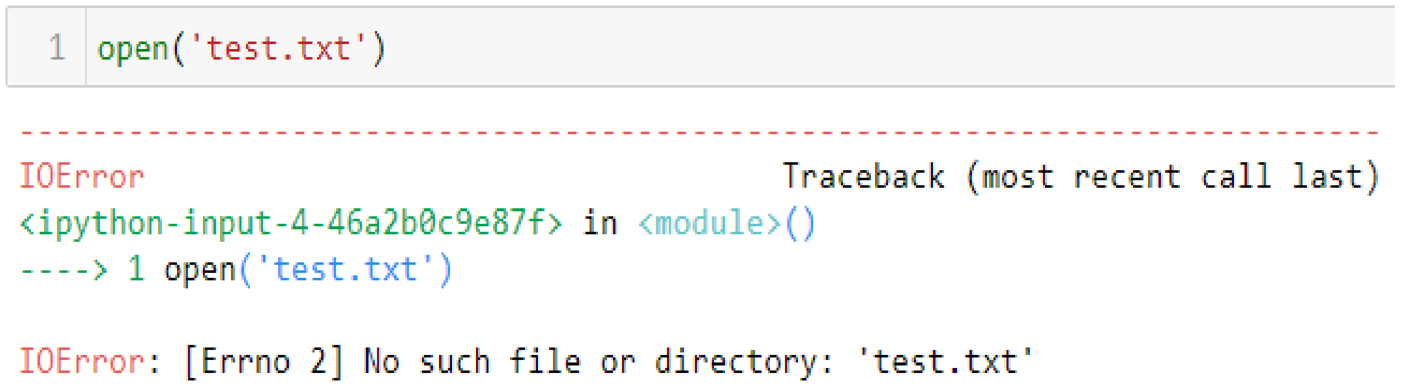
- Errors can also occur at runtime and these are **called exceptions.**
- for example, when a file we try to open does not exist (**FileNotFoundError**), dividing a number by zero (**ZeroDivisionError**),
- module we try to import is not found (**ImportError**) etc.

Error and Exceptions



```
1 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-3-b710d87c980c> in <module>()
----> 1 1 / 0

ZeroDivisionError: integer division or modulo by zero
```

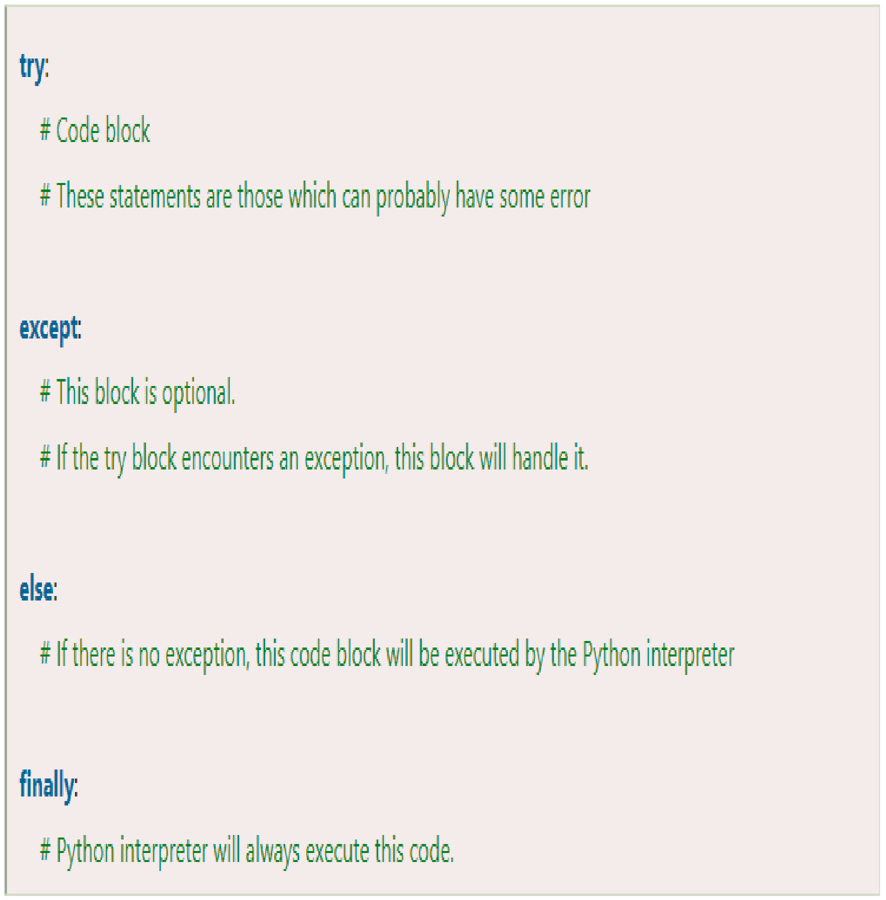


```
open('test.txt')
```

```
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
<ipython-input-4-46a2b0c9e87f> in <module>()
----> 1 open('test.txt')

IOError: [Errno 2] No such file or directory: 'test.txt'
```

Exception Handling

- If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.
- Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them –
  - Exception Handling –
  - Assertions – An assertion is a sanity-check that you can turn on or turn off when you are done with your testing of the program.



```
try:
  # Code block
  # These statements are those which can probably have some error

except:
  # This block is optional.
  # If the try block encounters an exception, this block will handle it.

else:
  # If there is no exception, this code block will be executed by the Python interpreter

finally:
  # Python interpreter will always execute this code.
```

Python Exception Handling

Try, EXcept and Finally

- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.
- When these exceptions occur, **it causes the current process to stop and passes it to the calling process until it is handled**. If not handled, our program will crash.

For example, if function A calls function B which in turn calls function C and an exception occurs in function C. **If it is not handled in C, the exception passes to B and then to A.**

If never handled, an error message is spit out and our program come to a sudden, unexpected halt.

Python Exception Handling

Catching Exceptions in Python

- In Python, exceptions can be handled **using a try statement**.
- **A critical operation which can raise exception is placed inside the try clause** and the code that handles exception is written in except clause.

Python Exception Handling

Catching Exceptions in Python

```
# import module sys to get the type of exception
import sys
lst = ['b', 0, 2]

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

Python Exception Handling

In the previous example, we **did not mention any exception in** the except clause.

We can specify which exceptions an except clause will catch.

A try clause can have <u>any number of except clause</u> to *handle them differently*

but only one will be executed in case an exception occurs.

Python Exception Handling

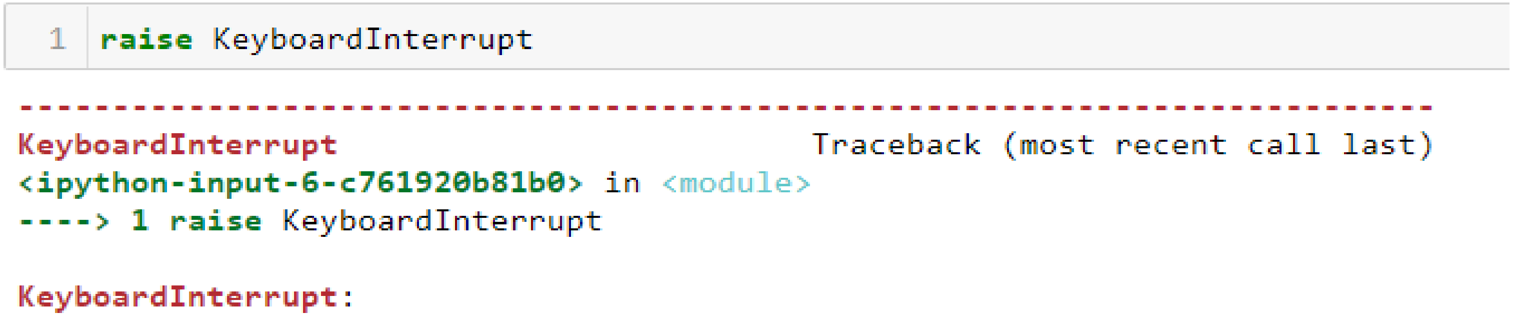
```
import sys
lst = ['b', 0, 2]

for entry in lst:
    try:
        print("*******************************")
        print("The entry is", entry)
        r = 1 / int(entry)
    except(ValueError):
        print("This is a ValueError.")
    except(ZeroDivisionError):
        print("This is a ZeroError.")
    except:
        print("Some other error")
print("The reciprocal of", entry, "is", r)
```

Raising Exceptions

- In Python programming, **exceptions are raised when corresponding errors occur at run time**, but we can **forcefully raise it using the keyword raise.**

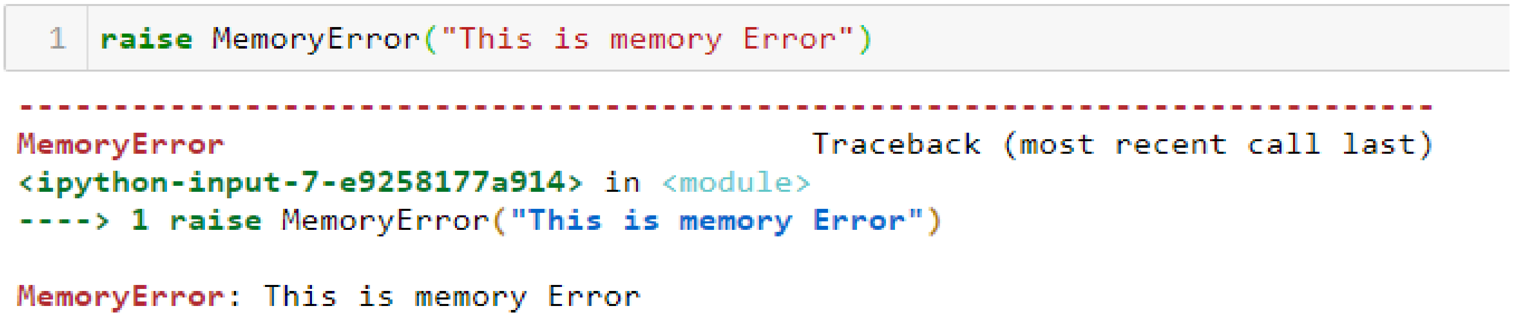
- We can also optionally pass in value to the exception to clarify why that exception was raised.

Raising Exceptions



```
raise KeyboardInterrupt
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-6-c761920b81b0> in <module>
----> 1 raise KeyboardInterrupt

KeyboardInterrupt: 
```



```
raise MemoryError("This is memory Error")
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-7-e9258177a914> in <module>
----> 1 raise MemoryError("This is memory Error")

MemoryError: This is memory Error
```

Exceptions Handling

try ... finally

The try statement in Python can have an optional finally clause.

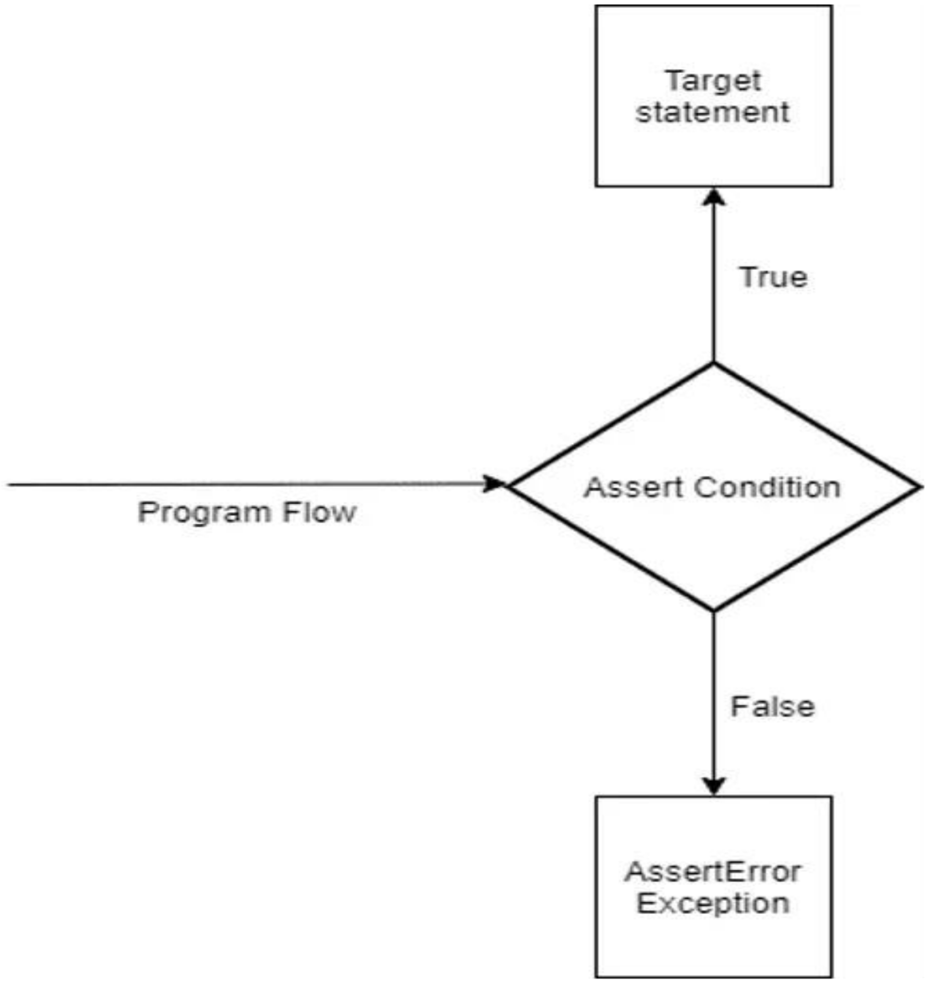
This clause is executed no matter what, and is generally used to release external resources.

```
try:
    f = open("sample1.txt", "r")
    print(f.readline())

finally:
    f.close()
```

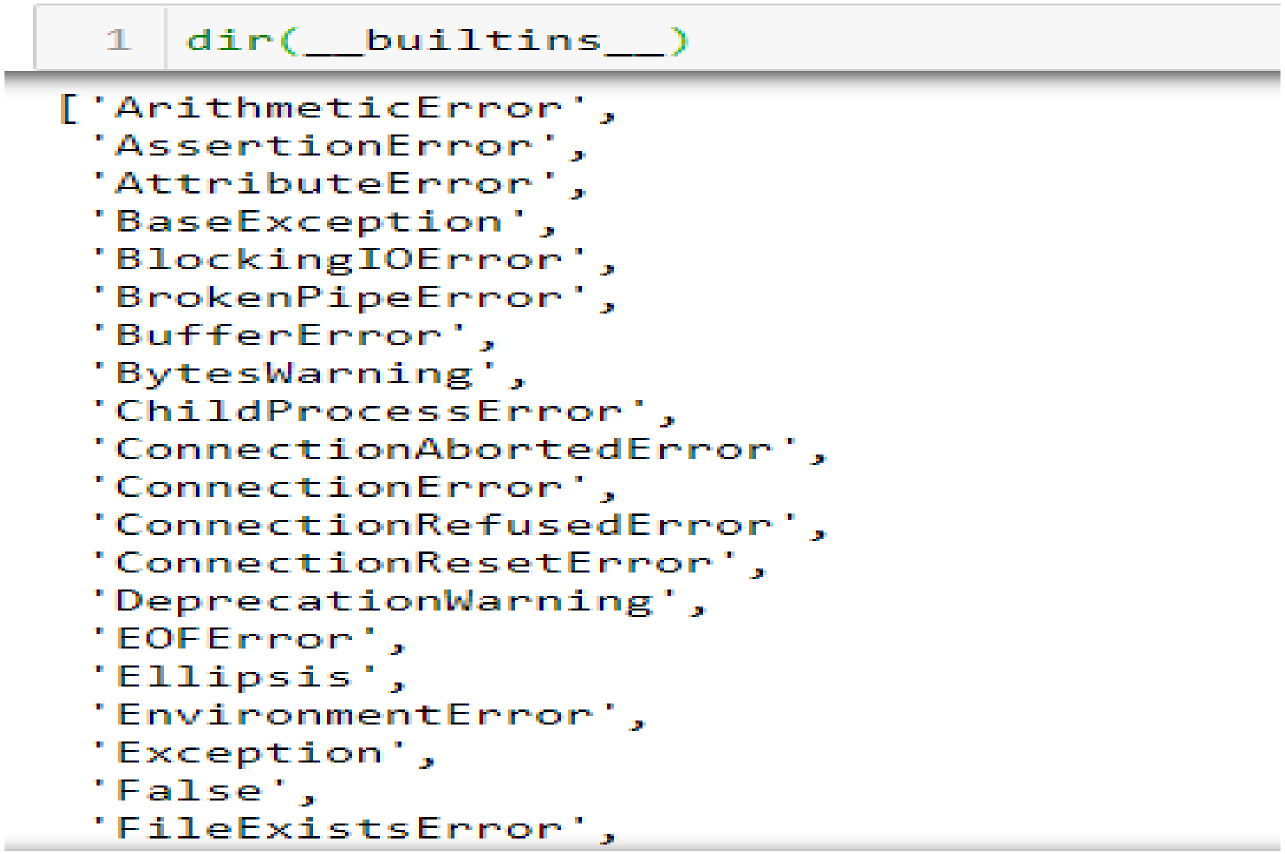
built-in exception

there are several built-in exceptions like:

1. ModuleNotFound Error
2. ImportError
3. MemoryError
4. OSError
5. SystemError



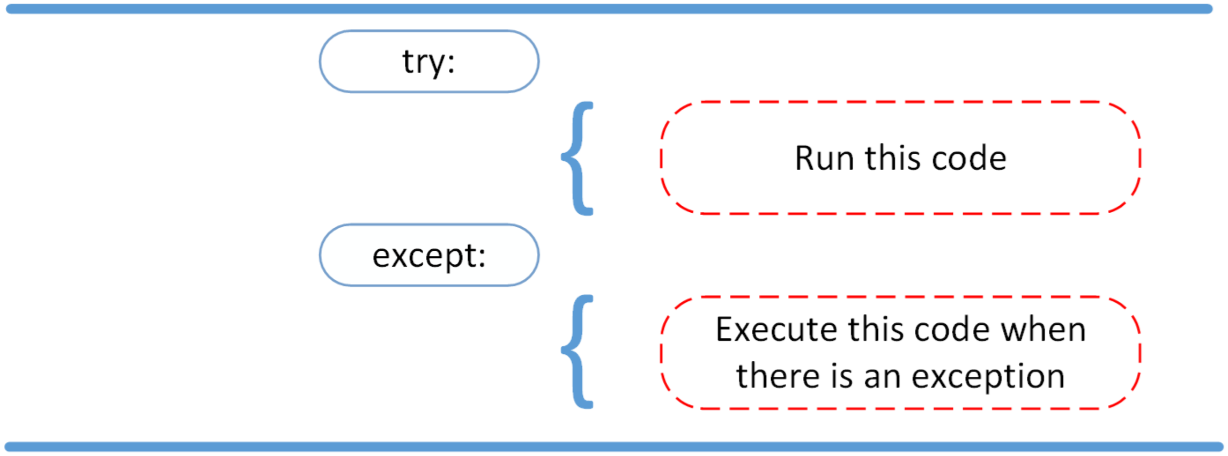
Python Built-in Exceptions

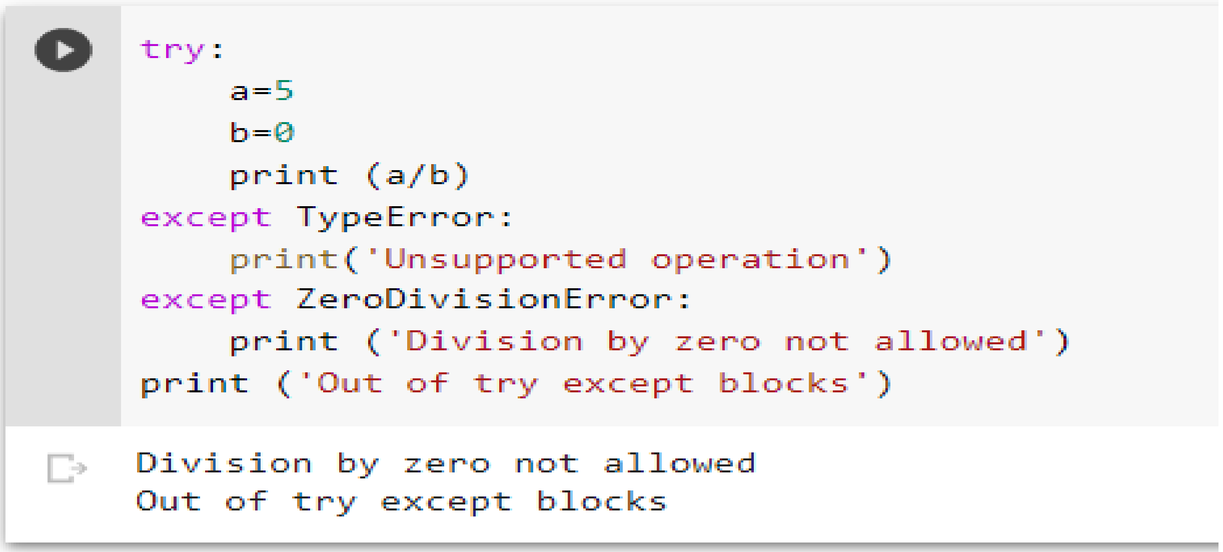


```
dir(__builtins__)
```

```
['ArithmeticError',
 'AssertionError',
 'AttributeError',
 'BaseException',
 'BlockingIOError',
 'BrokenPipeError',
 'BufferError',
 'BytesWarning',
 'ChildProcessError',
 'ConnectionAbortedError',
 'ConnectionError',
 'ConnectionRefusedError',
 'ConnectionResetError',
 'DeprecationWarning',
 'EOFError',
 'Ellipsis',
 'EnvironmentError',
 'Exception',
 'False',
 'FileExistsError',
```

Exception Handling





```
try:
    a=5
    b=0
    print (a/b)
except TypeError:
    print('Unsupported operation')
except ZeroDivisionError:
    print ('Division by zero not allowed')
print ('Out of try except blocks')
```

```
Division by zero not allowed
Out of try except blocks
```

References

- https://www.javatpoint.com/python-features
- https://www.w3schools.com/python/python_intro.asp
- https://www.geeksforgeeks.org/python-data-types/
- https://www.tutorialsteacher.com/python/python-data-types
- https://www.programiz.com/python-programming/variables-datatypes
- http://sthurlow.com/python/lesson06/
- https://subscription.packtpub.com/book/application_development/9781789800111/1/ch01lvl1sec04/lists-sets-strings-tuples-and-dictionaries
- https://www.edureka.co/blog/data-structures-in-python/
- https://www.guru99.com/if-loop-python-conditional-structures.html
- https://beginnersbook.com/2018/01/python-functions/
- Python Function Exercises with Solution (pythonlobby.com)