Python programming for beginners

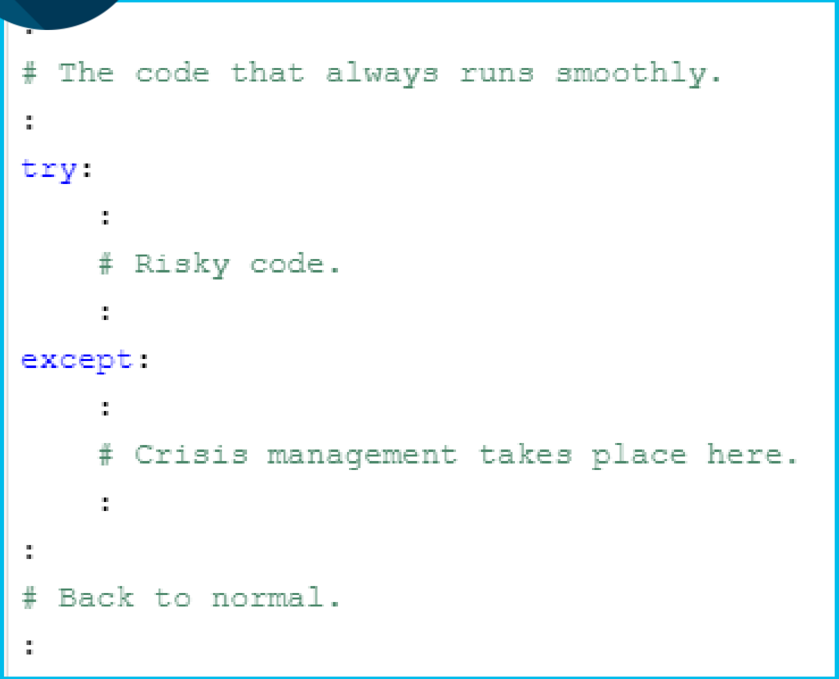
Stefan Zhauryd
Instructor

Module 6

Exceptions

In this module, you will learn about:

- Python's way of handling runtime errors;
- Controlling the flow of errors using try and except;
- Hierarchy of exceptions.



```
:
# The code that always runs smoothly.
:
try:
    :
    # Risky code.
    :
except:
    :
    # Crisis management takes place here.
    :
:
# Back to normal.
:
```

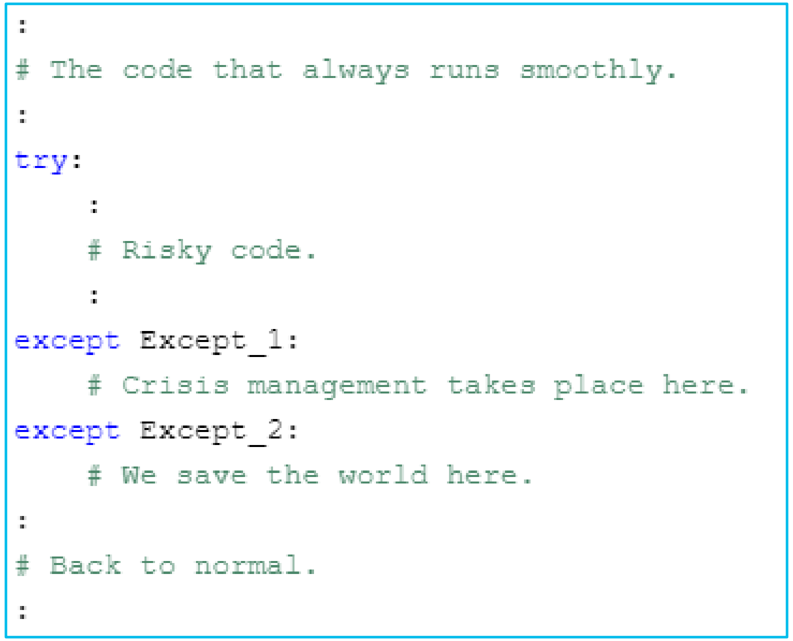
(ZeroDevErr, dsvl)

A = 9

If dvsdv == 9:

elif sdsddf:

elif dsfsdf:



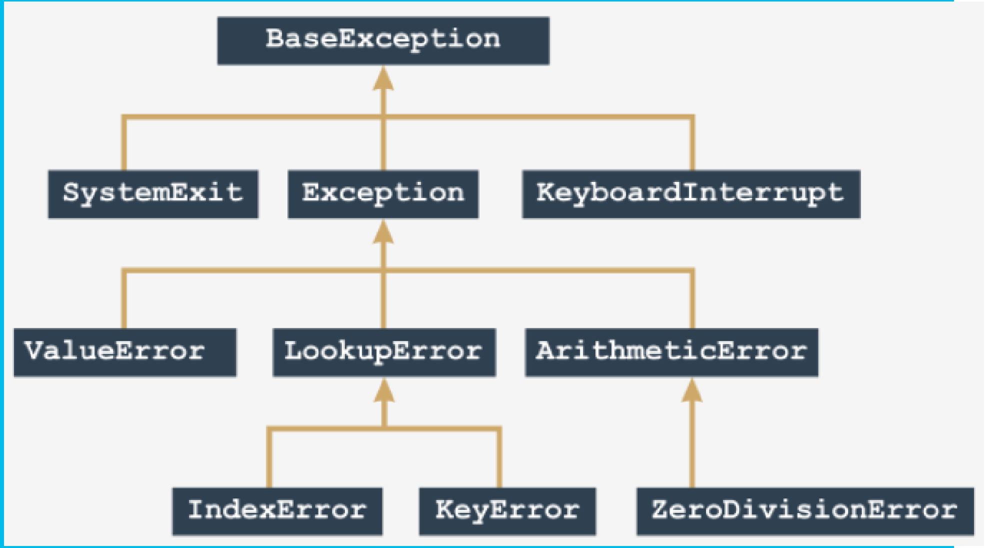
```
:
# The code that always runs smoothly.
:
try:
    :
    # Risky code.
    :
except Except_1:
    # Crisis management takes place here.
except Except_2:
    # We save the world here.
:
# Back to normal.
:
```

```
# The code that always runs smoothly.
:
try:
    :
    # Risky code.
    :
except Except_1:
    # Crisis management takes place here.
except Except_2:
    # We save the world here.
except:
    # All other issues fall here.
:
# Back to normal.
:
```

```
try:
    print("Let's try to do this")
    print("#"[2])
    print("We succeeded!")
except:
    print("We failed")
print("We're done")
```

```
try:
    print("alpha"[1/0])
except ZeroDivisionError:
    print("zero")
except IndexingError:
    print("index")
except:
    print("some")
```

Exceptions

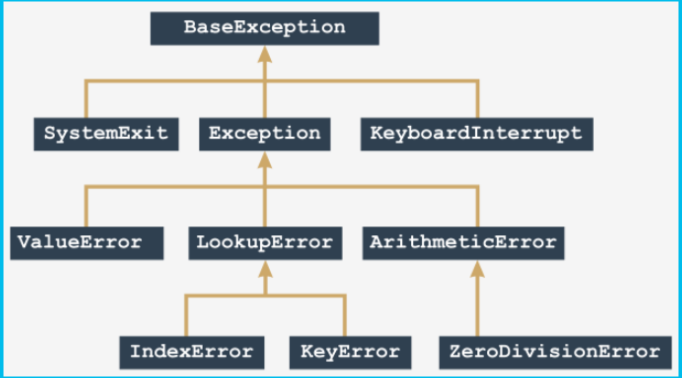


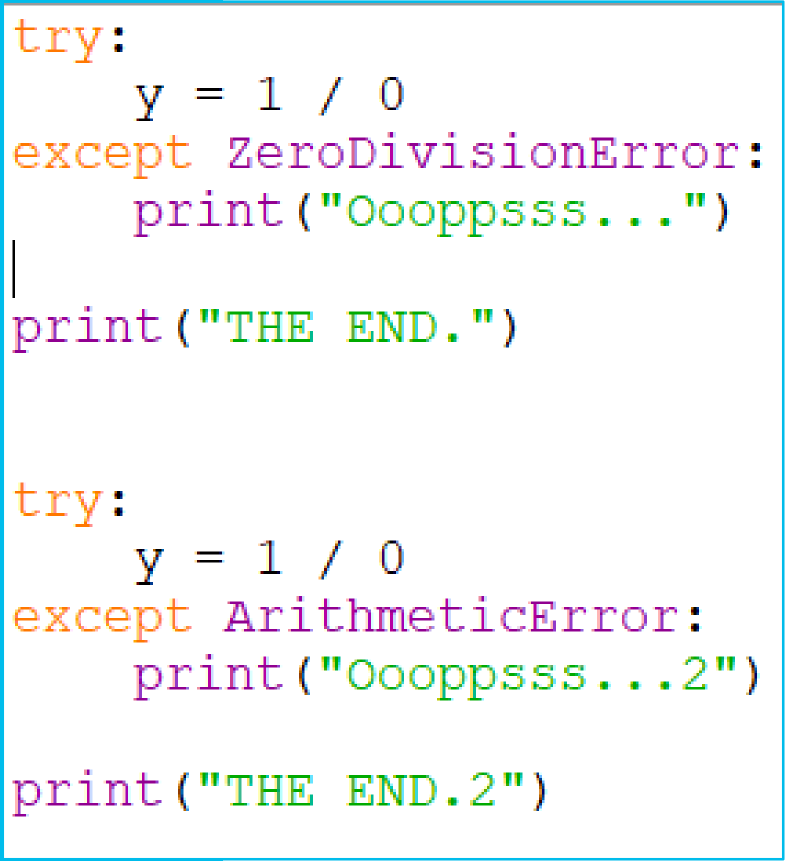
Note:

- **ZeroDivisionError** is a special case of more a general exception class named **ArithmeticError**;
- **ArithmeticError** is a special case of a more general exception class named just **Exception**;
- **Exception** is a special case of a more general class named **BaseException**;

https://docs.python.org/3/tutorial/errors.html

Exceptions: continued





```
try:
    y = 1 / 0
except ZeroDivisionError:
    print("Oooppsss...")

print("THE END.")



try:
    y = 1 / 0
except ArithmeticError:
    print("Oooppsss...2")

print("THE END.2")
```

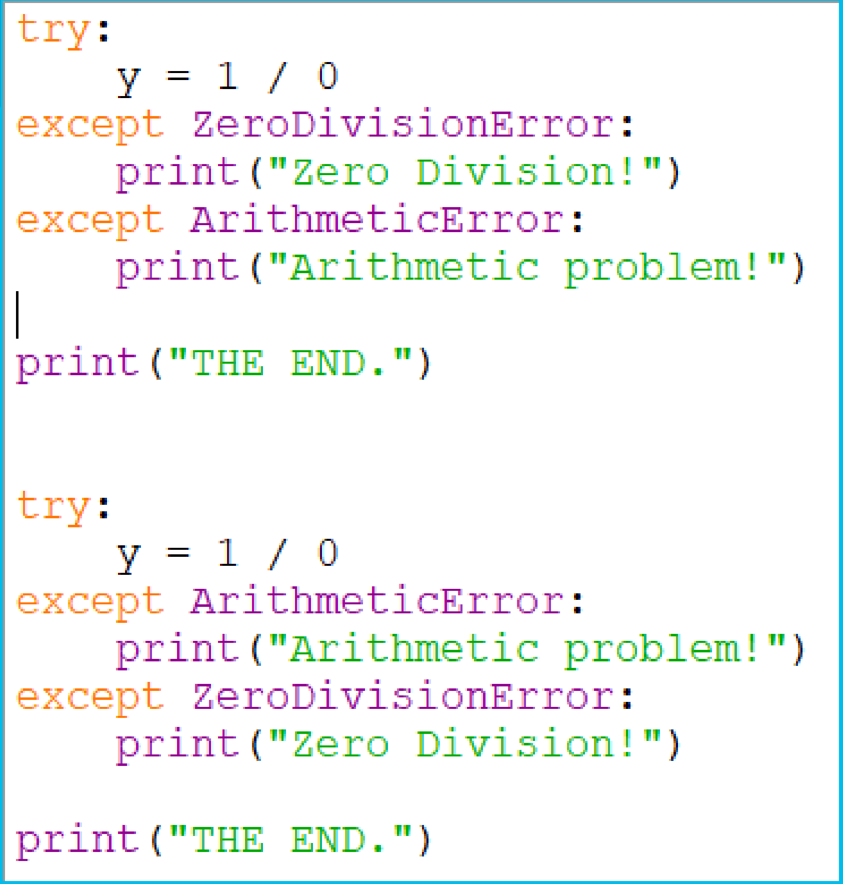
Let’s try to change on:

BaseException

Or

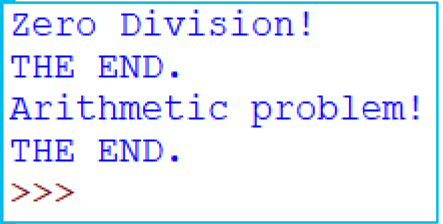
Exception

```
Oooppsss...
THE END.
Oooppsss...2
THE END.2
>>>
```

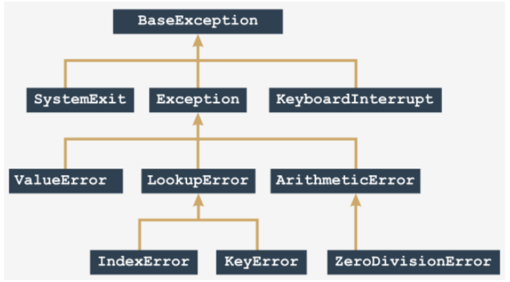


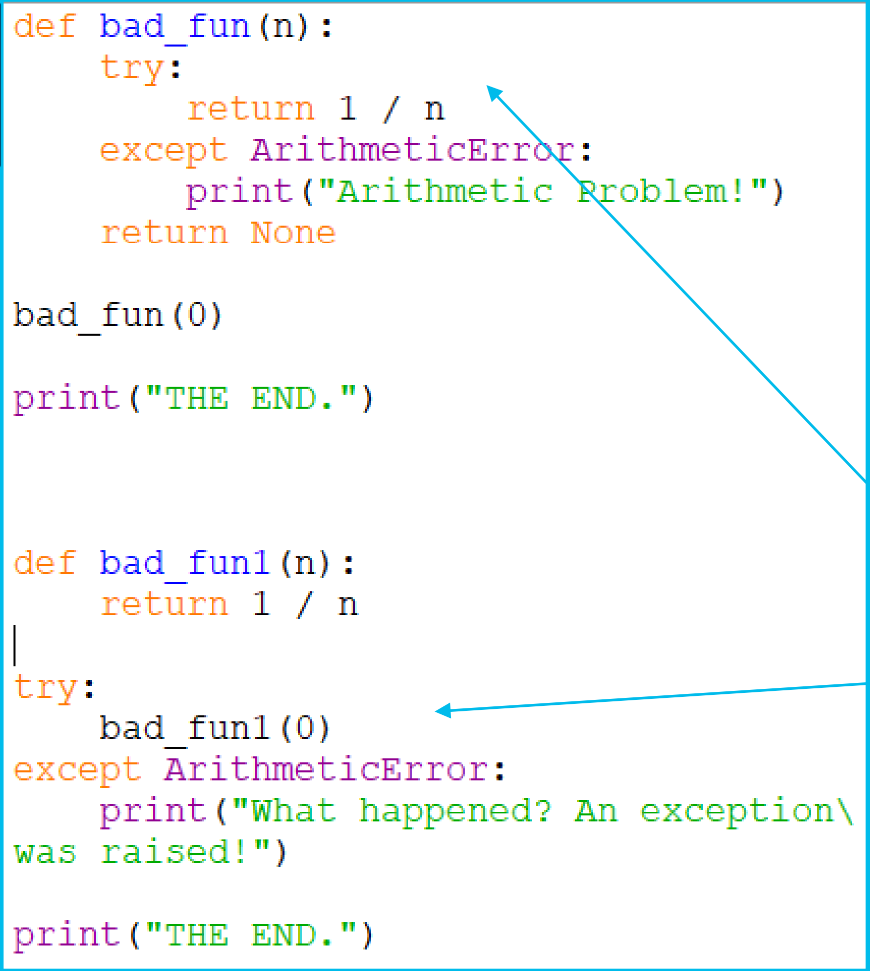
```
try:
    y = 1 / 0
except ZeroDivisionError:
    print("Zero Division!")
except ArithmeticError:
    print("Arithmetic problem!")

print("THE END.")


try:
    y = 1 / 0
except ArithmeticError:
    print("Arithmetic problem!")
except ZeroDivisionError:
    print("Zero Division!")

print("THE END.")
```

- The order of the branches matters!
- Do not put more general exceptions before more specific ones;
- This will make the latter inaccessible and useless;
- Moreover, it will make your code messy and inconsistent;
- Python will not generate any error messages on this issue.

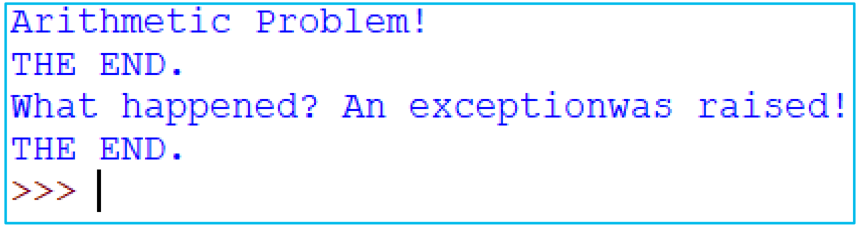


```
Zero Division!
THE END.
Arithmetic problem!
THE END.
>>>
```





```
def bad_fun(n):
    try:
        return 1 / n
    except ArithmeticError:
        print("Arithmetic Problem!")
    return None

bad_fun(0)

print("THE END.")



def bad_fun1(n):
    return 1 / n

try:
    bad_fun1(0)
except ArithmeticError:
    print("What happened? An exception\
was raised!")

print("THE END.")
```

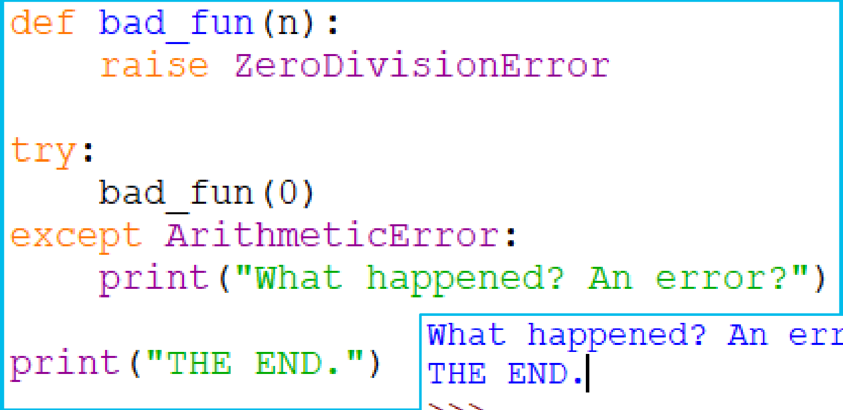


```
Arithmetic Problem!
THE END.
What happened? An exceptionwas raised!
THE END.
>>> 
```

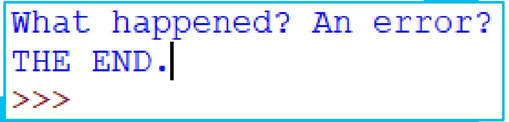
If an exception is raised inside a function, it can be handled:

- **inside the function;**
- **outside the function;**

Exceptions: continued



```
def bad_fun(n):
    raise ZeroDivisionError

try:
    bad_fun(0)
except ArithmeticError:
    print("What happened? An error?")

print("THE END.")
```



```
What happened? An error?
THE END.
>>>
```

The **raise** instruction **raises** the specified exception named **exc** as if it was raised in a normal (natural) way:

```
raise exc
```

Note: **raise is a keyword.**

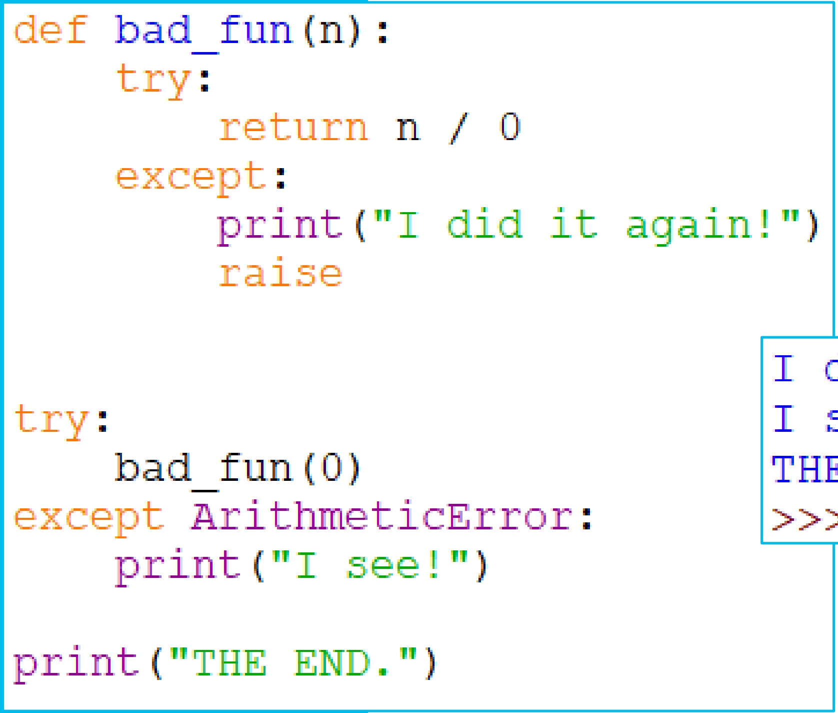
The instruction enables you to:

- **simulate raising actual exceptions** (e.g., to test your handling strategy)
- partially **handle an exception** and make another part of the code responsible for completing the handling (separation of concerns).

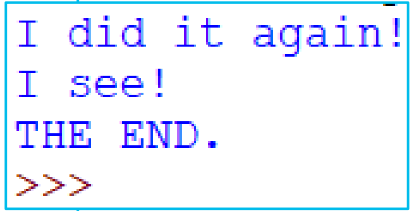
Exceptions: continued

```
raise
```

this kind of raise instruction may be used inside the except branch only; using it in any other context causes an error.

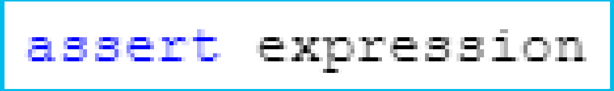


```
def bad_fun(n):
    try:
        return n / 0
    except:
        print("I did it again!")
        raise


try:
    bad_fun(0)
except ArithmeticError:
    print("I see!")

print("THE END.")
```



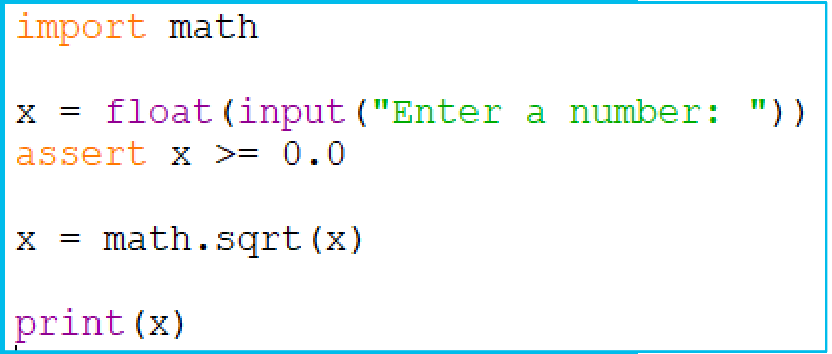
```
I did it again!
I see!
THE END.
>>>
```

Exceptions: continued

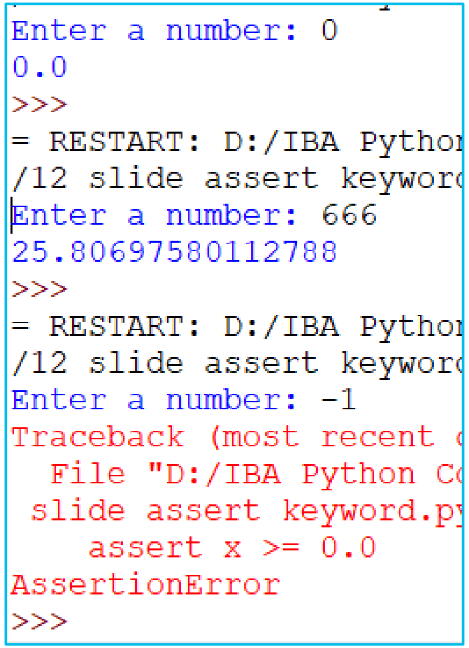


```
assert expression
```

evaluates the expression and **raises the AssertionError exception when the expression is equal to zero, an empty string, or None, or False**.



```
import math

x = float(input("Enter a number: "))
assert x >= 0.0

x = math.sqrt(x)

print(x)
```



```
Enter a number: 0
0.0
>>>
= RESTART: D:/IBA Pythor
/12 slide assert keyworc
Enter a number: 666
25.80697580112788
>>>
= RESTART: D:/IBA Pythor
/12 slide assert keyworc
Enter a number: -1
Traceback (most recent c
  File "D:/IBA Python Co
 slide assert keyword.py
    assert x >= 0.0
AssertionError
>>>
```

Key takeaways

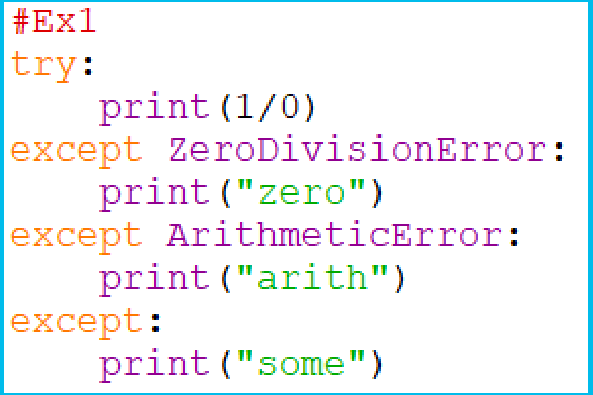
The Python statement:

raise *ExceptionName* - can raise an exception on demand. The same statement, but lacking ExceptionName, can be used inside the try branch only, and raises the same exception which is currently being handled.

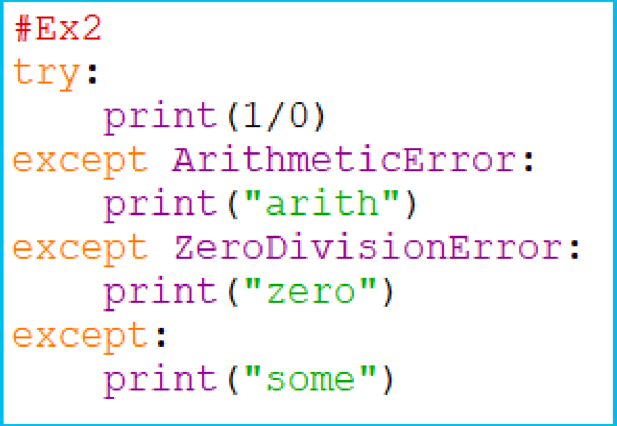
The Python statement:

assert *expression* - evaluates the expression and **raises the AssertionError exception when the expression is equal to zero, an empty string, or None, or False**. You can use it to protect some critical parts of your code from devastating data.

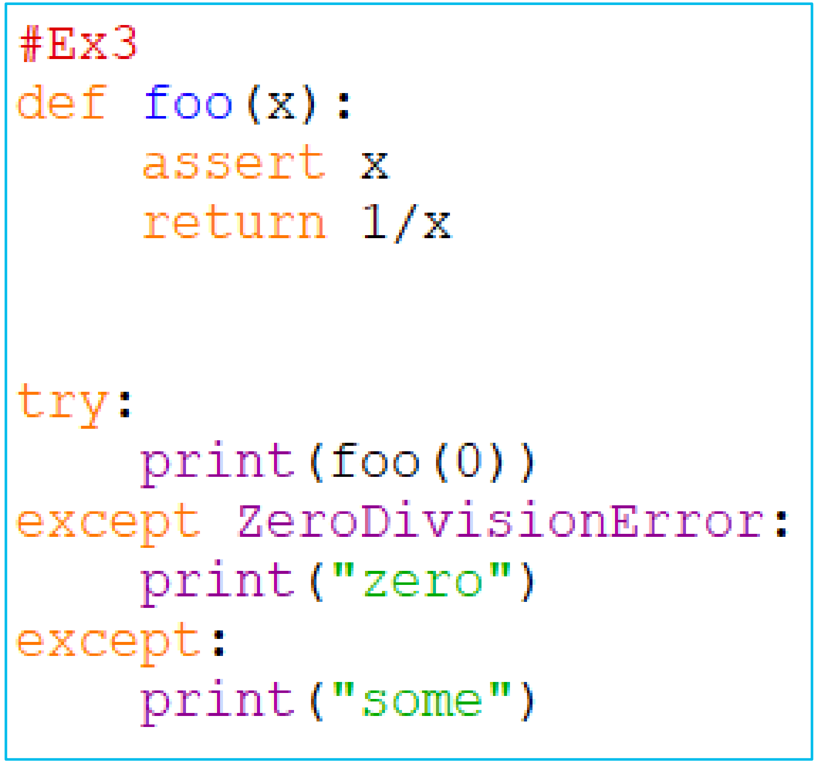
Examples



```
#Ex1
try:
    print(1/0)
except ZeroDivisionError:
    print("zero")
except ArithmeticError:
    print("arith")
except:
    print("some")
```



```
#Ex2
try:
    print(1/0)
except ArithmeticError:
    print("arith")
except ZeroDivisionError:
    print("zero")
except:
    print("some")
```



```
#Ex3
def foo(x):
    assert x
    return 1/x


try:
    print(foo(0))
except ZeroDivisionError:
    print("zero")
except:
    print("some")
```

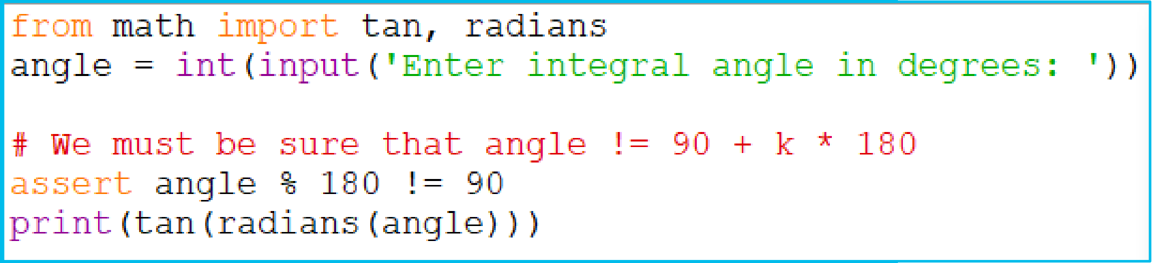
AssertionError

Location:

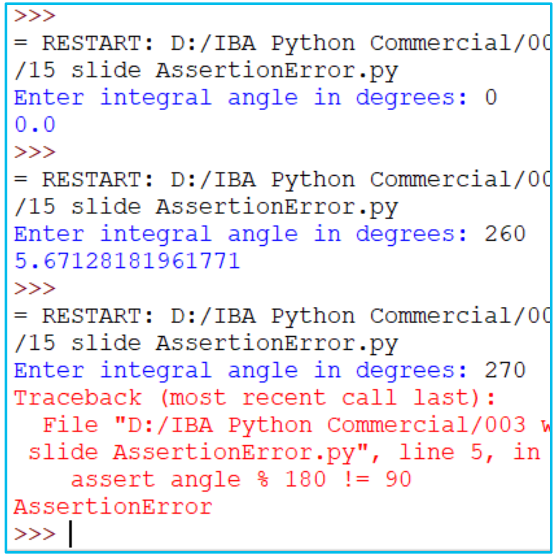
- BaseException ← Exception ← AssertionError

Description:

- a **concrete exception** raised by the assert instruction **when its argument evaluates to False, None, 0, or an empty string**

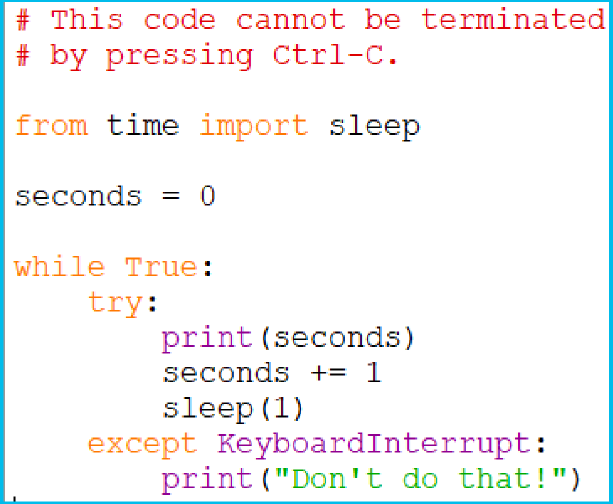


```python
from math import tan, radians
angle = int(input('Enter integral angle in degrees: '))

# We must be sure that angle != 90 + k * 180
assert angle % 180 != 90
print(tan(radians(angle)))
```



```
>>>
= RESTART: D:/IBA Python Commercial/00
/15 slide AssertionError.py
Enter integral angle in degrees: 0
0.0
>>>
= RESTART: D:/IBA Python Commercial/00
/15 slide AssertionError.py
Enter integral angle in degrees: 260
5.67128181961771
>>>
= RESTART: D:/IBA Python Commercial/00
/15 slide AssertionError.py
Enter integral angle in degrees: 270
Traceback (most recent call last):
  File "D:/IBA Python Commercial/003 w
 slide AssertionError.py", line 5, in
    assert angle % 180 != 90
AssertionError
>>> |
```

KeyboardInterrupt



```
# This code cannot be terminated
# by pressing Ctrl-C.

from time import sleep

seconds = 0

while True:
    try:
        print(seconds)
        seconds += 1
        sleep(1)
    except KeyboardInterrupt:
        print("Don't do that!")
```

Location:

- BaseException ← KeyboardInterrupt

Description:

- a **concrete exception** raised when the user uses a keyboard shortcut designed to terminate a program's execution **(Ctrl-C in most OSs**); if handling this exception doesn't lead to program termination, the program continues its execution.

Note: this exception is not derived from the Exception class. Run the program in IDLE.

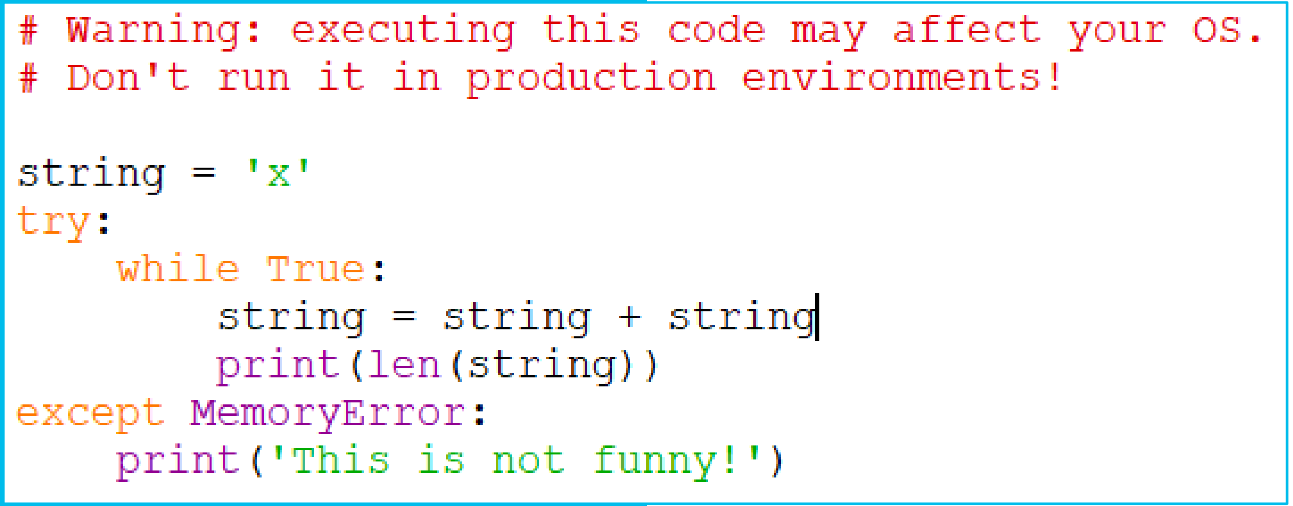
MemoryError

Location:

BaseException ← Exception ← MemoryError

Description:

a **concrete exception** raised when an operation cannot be completed due to a lack of free memory.



```python
# Warning: executing this code may affect your OS.
# Don't run it in production environments!

string = 'x'
try:
    while True:
        string = string + string
        print(len(string))
except MemoryError:
    print('This is not funny!')
```

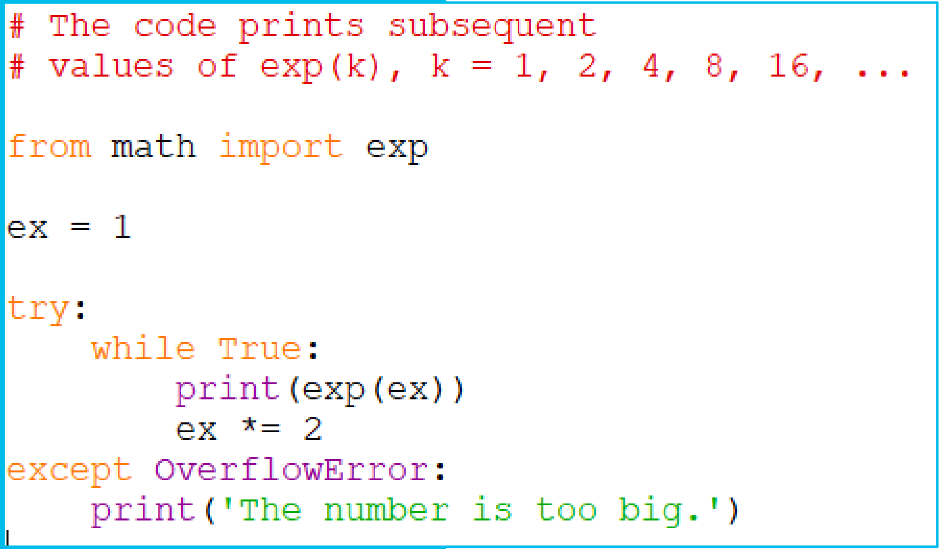
OverflowError

Location:

BaseException ← Exception ← ArithmeticError ← OverflowError

Description:

a **concrete exception** raised when an operation produces a number too big to be successfully stored



```python
# The code prints subsequent
# values of exp(k), k = 1, 2, 4, 8, 16, ...

from math import exp

ex = 1

try:
    while True:
        print(exp(ex))
        ex *= 2
except OverflowError:
    print('The number is too big.')
```

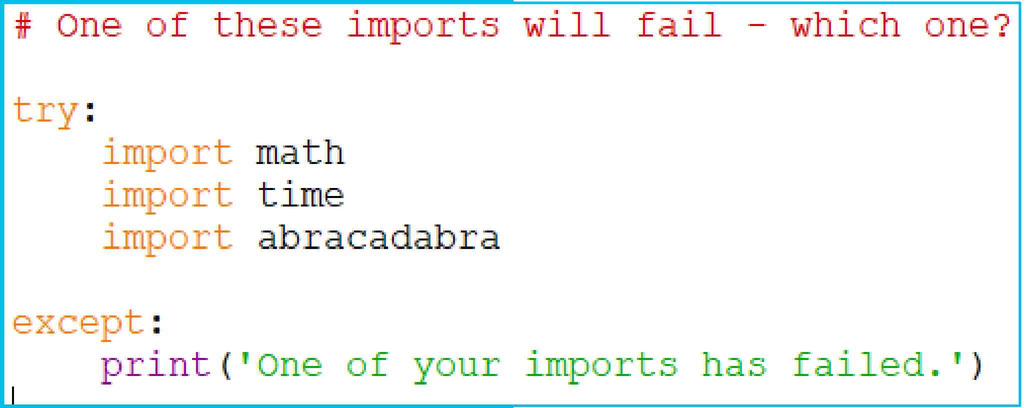
ImportError

Location:

BaseException ← Exception ← StandardError ← ImportError

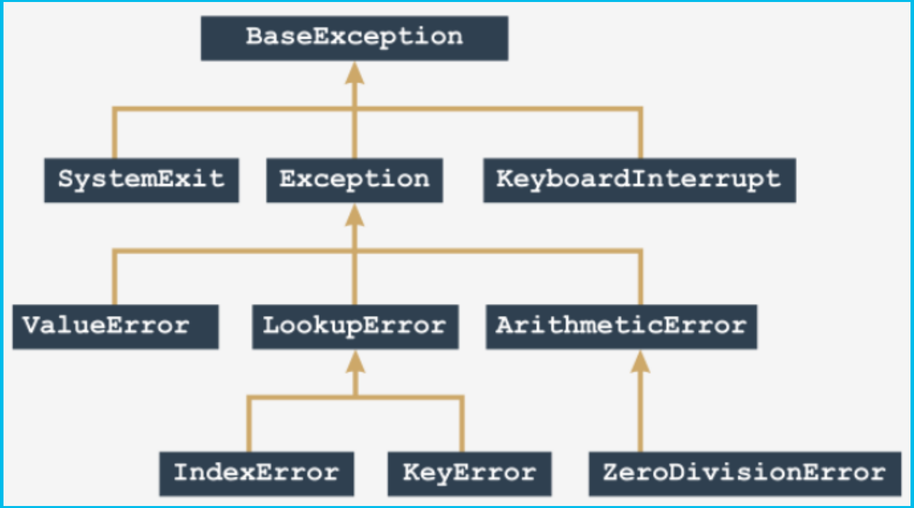
Description:

a **concrete exception** raised when an import operation fails



```
# One of these imports will fail - which one?

try:
    import math
    import time
    import abracadabra

except:
    print('One of your imports has failed.')
```

Well done



For the time being, if you'd like to learn more about exceptions on your own, you look into Standard Python Library at https://docs.python.org/3.9/library/exceptions.html

1. Some **abstract built-in** Python exceptions are:
ArithmeticError,
BaseException,
LookupError.

2. Some **concrete built-in** Python exceptions are:
AssertionError,
ImportError,
IndexError,
KeyboardInterrupt,
KeyError,
MemoryError,
OverflowError.

ЗАДАНИЯ

1) Прорешать всю классную работу

Почитать:

1) Byte of Python

**) Structuring Your Project:

Крайний срок сдачи 14/10 в 21:00 (можно раньше, но не позже)

ЗАДАНИЯ

Название файлов, которые вы отправляете мне в telegram:
Vasia_Pupkin_class_work_Exception_L9_P0.py

Формат сообщения которое вы присылаете мне
(после полного выполнения домашнего задания, только один раз) в Telegram:
Добрый день/вечер. Я Вася Пупкин, и это мои домашние задания к лекции 9 часть 0 про исключения.
И отправляете файл/-лы

Крайний срок сдачи 14/10 в 21:00 (можно раньше, но не позже)

https://docs.github.com/articles/using-pull-requests

Q&A

Create your
possibilities.
Bye bye.