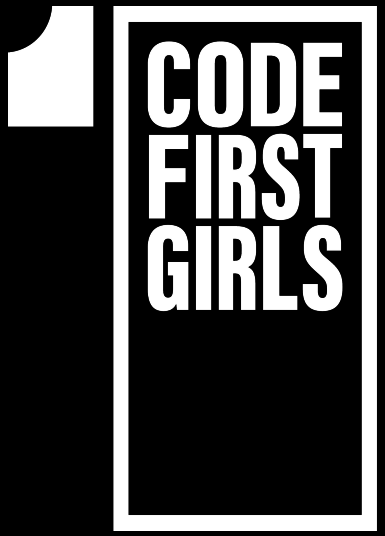
# EXCEPTION & DEBUGGING

## LESSON 8




CFGDEGREE → FOUNDATION MODULE

# AGENDA

01 Introduction to Exception & Debugging

02 Assertion

03 Exceptions and Errors

04 Exception Handling

05 User Defined Exceptions

06 Debugging

07 Examples and Exercises

# INTRODUCTION

## EXCEPTION AND DEBUGGING

Python provides very important features to handle any **unexpected behaviour** or **error** in your programs, as well as add debugging capabilities in them:

- Exception Handling
- Assertions
- Breakpoints for debugging

#EXAMPLE

```
def add_vat(vat, prices):
    """
    Add commission to every price item in the provided iterable.

    :param vat: float, vat percentage
    :param prices: iterable, net prices as per customers' receipt
    :return: list of prices with added vat
    """
    new_prices = [(price / 100 * vat) + price for price in prices]

    return new_prices
```

```
In[18]: add_vat(vat=20, prices=[24, 0.15, '10', 32.45])
Traceback (most recent call last):
  File "C:\Users\olyan\AppData\Local\Programs\Python\Python38-3
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-18-b10b7eb1a487>", line 1, in <module>
    add_vat(vat=20, prices=[24, 0.15, '10', 32.45])
```

# ASSERTION

## HOW IT WORKS

- An assertion is a sanity-check that we can turn on or turn off when we finished testing of the program.
- Think of it as a **raise-if** statement (or to be more accurate, a raise-if-not statement).
- Assertions are carried out by the assert statement
- Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output

#SYNTAX

**assert Expression[, Arguments]**

```
def apply_discount(product, discount):
    """
    Add a discount to the price.
    :param product: dict obj, item spec including price
    :param discount: float discount expressed in percent
    :return: float new price
    """
    price = product['price'] * (1.0 - (discount / 100))
    assert 0 <= price <= product['price']
    return price
```

# ASSERTION

## CAVEATS

### #1 Do not Use Asserts for Data Validation

- Asserts can be turned off globally in the Python interpreter
- Disabled assert statements turn into a null-operation: the assertions simply get compiled away and won't be evaluated

#EXAMPLE

**Very Bad Practice**

```
def cancel_membership(membership_id, user):
    """
    Canel Gym membership for an existing member.
    """
    assert user.is_admin(), 'Must be admin to cancel'
    assert gym_members.membership_exists(membership_id), 'Unknown id'
    gym_members.find_membership(membership_id).delete()
```

# ASSERTION

## CAVEATS

### #1 Do not Use Asserts for Data Validation

- Asserts can be turned off globally in the Python interpreter
- Disabled assert statements turn into a null-operation: the assertions simply get compiled away and won't be evaluated

#EXAMPLE

**How to avoid this**

```
def cancel_membership(membership_id, user):
    """
    Canel Gym membership for an existing member.
    """
    if not user.is_admin():
        raise AuthorisationError('Must be admin to cancel')
    if not gym_members.membership_exists(membership_id):
        raise ValueError('Unknown id')

    gym_members.find_membership(membership_id).delete()
```

# ASSERTION

## CAVEATS

### #2 Assert That Never Fails

- When you pass a **tuple as the first argument** in an assert statement, the assertion always evaluates as true
- This kind of assertion will never fail.

#EXAMPLE

**Never Pass a Tuple to Assert**

```
assert(1 == 2, 'This should fail')
```

**But it does not fail!**

# HANDLING EXCEPTIONS

## HOW IT WORKS

- An *exception* is a Python object that represents an *error.*
- When a Python script encounters a situation that it cannot cope with, it raises an exception.
- When an exception is raised, it must either be *handled immediately*, otherwise it breaks the flow, *terminates and quits*.

#EXAMPLE

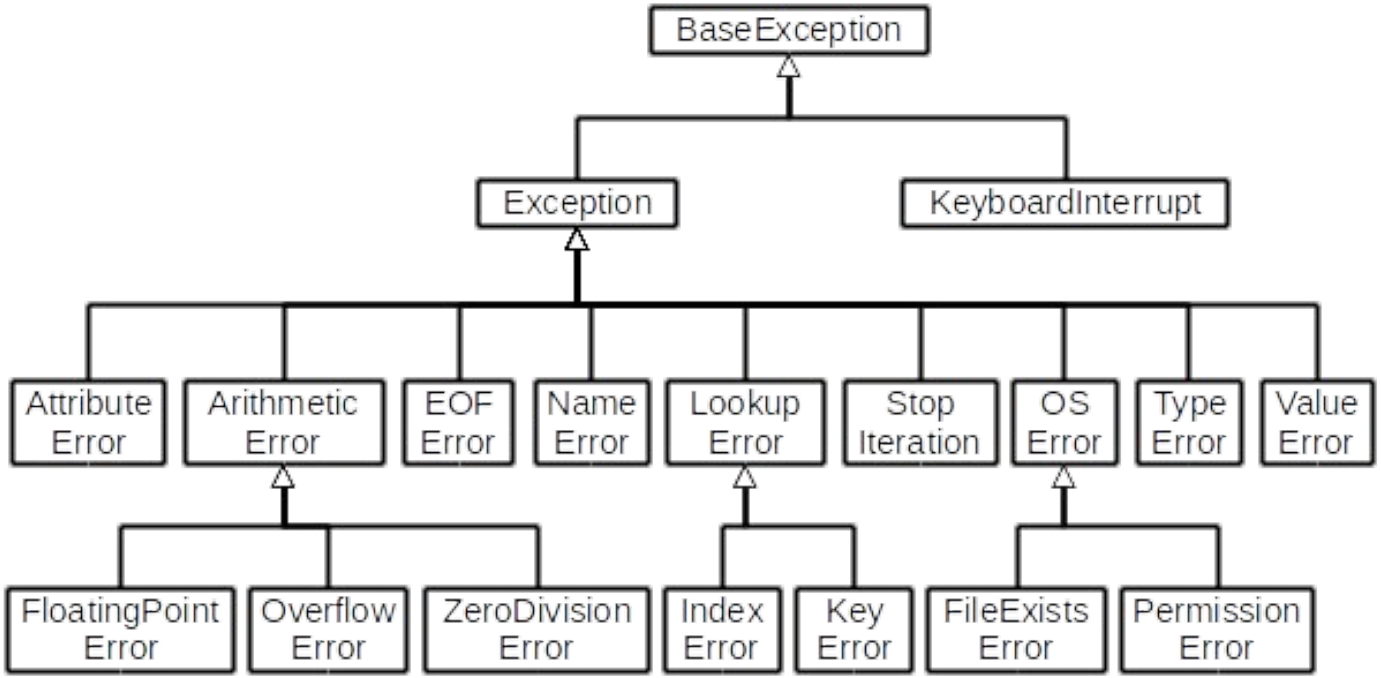
**Exception Parent Object**

```
In[5]: help(Exception)
Help on class Exception in module builtins:

class Exception(BaseException)
 |  Common base class for all non-exit exceptions.
 |
 |  Method resolution order:
 |      Exception
 |      BaseException
 |      object
```

# EXCEPTION TYPES

## HIERARCHY



# BUILT IN ERRORS & EXCEPTIONS

## HOW IT WORKS

- A python program terminates as soon as it encounters an unhandled error. These errors can be broadly classified into two categories:
-
    1. Syntax errors
    2. Logical errors (Exceptions)

#EXAMPLE

**Error caused by not following the proper structure (syntax) of the language**

```
>>> if x < 5
  File "<interactive input>", line 1
                 if x < 5
                      ^
  SyntaxError: invalid syntax
```

**Errors that occur at runtime (after passing the syntax test)**

https://docs.python.org/3/library/exceptions.html

# EXCEPTION & ERROR HANDLING

## IN PYTHON

- Exception and Error handling increases the robustness of your code

- It guards against potential failures that would cause your program to exit in an uncontrolled fashion.

#EXAMPLE

**Never Pass a Tuple to Assert**

```
assert(1 == 2, 'This should fail')
```

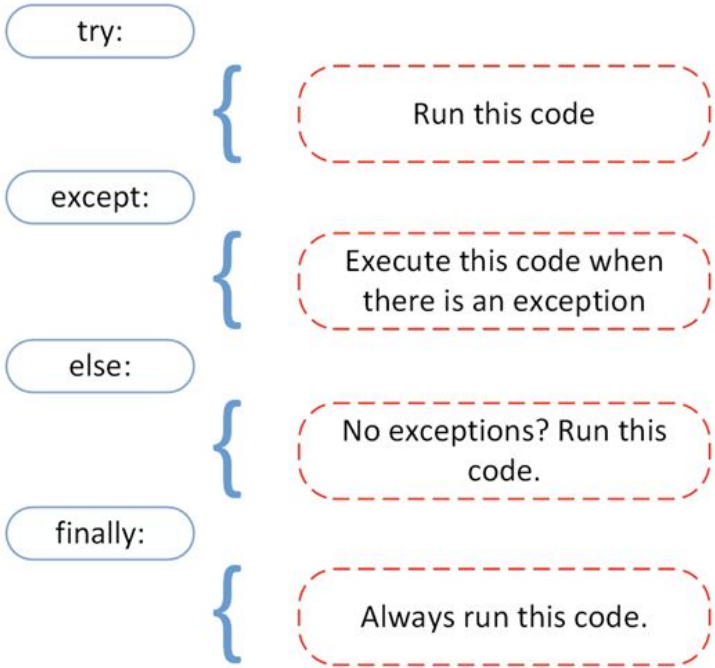
**But it does not fail!**

# EXCEPTION HANDLING

## IN PYTHON

- Receiving the exception object and performing the necessary actions for dealing with the bug refers to handling the exception.
- This would be the analyzing and correcting steps in the debugging process.
- Manage the flow of your program with the **“try / except”** block

#IMPORTANT SYNTAX




**DEMO**

# RAISE KEYWORD

## IN PYTHON

- The **raise** keyword is another way to handle exceptions in Python.
- In this case, you will be able to raise your own exceptions
- These are exceptions that get raised when an issue outside the scope of expected errors occurs.

#EXAMPLE

```
number = int(input('Enter a number in the range 5-10: '))

try:
    if number < 5 or number > 10:
        raise Exception

    division_result = 100 / number
    print(division_result)
    print("Well Done")

except:
    print("Your number is NOT in the requested range")
```

**DEMO**

# USER DEFINED EXCEPTIONS

## IN PYTHON

- Python allows you to create your OWN exceptions.
- This is achieved by deriving (inheriting) classes from the standard built-in exceptions.

**NOTE: we have not learnt about classes or inheritance yet, so at this stage we only need to know that we CAN make our OWN exceptions.**

#EXAMPLE

```
class ValueIsBelowHundredError(ValueError):
    """Raised when value is below 100"""
    pass
```

# DEBUGGING

## DEFINITIONS

### BUG

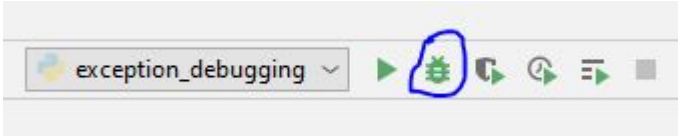
- Bugs in software cause the program to produce unintended behaviour. It is a term commonly used to refer to an *error* with unknown location and reason, which can cause severe issues (i.e. crashing a program).

### DEBUGGING

- Debugging is the process of locating, analyzing, and correcting any bug (error) you might encounter. The ultimate goal of debugging is to remove such bugs.

## #HOW TO DEBUG

**If using a 'clever' IDE e.g. PyCharm, we can use in-built debugging functionality to run a program in debugging mode.**




**When running a program with a simple terminal, we can use Python library called pdb to run programs in debugging mode.**

```
import pdb
pdb.set_trace()
```

# EXERCISES

## PRACTICE & CODING

WITH YOUR INSTRUCTOR

**Teenager club registration program:**

- Manage the program execution by validating user's name and age.
- Raise Validation and Assertion errors for invalid input.
- Write a registration record into a new file.

INDEPENDENTLY

**Write a function that can read contents of a file and can handle cases when provided file name does not exist:**

- Handle Error cases gracefully displaying an informative message to the user.
- What Error type can we use here? (check Python documentation)

# THANK YOU!