Python | Main course

Session 13

Exceptions

Assertion

Exception Handling

by Mohammad Amin H.B. Tehrani

www.maktabsharif.ir

Introduction

Let's start with an Example...

Division function

1. Simple Division function

```
>> def division(a, b):
>>     return a / b
```

```
>> division(10, 5)  # Works fine

Output:
>> 2.0
```

Division function

```
>> division(10, 0)   # ?
```

```
Output:
zeroDivisionError
Traceback (most recent call last)
<ipython-input-15-658ea34c5191> in <module>()
----> 1 division(10, 0)  # ?

<ipython-input-13-d012b66df33b> in division(a, b)
      1 def division(a, b):
----> 2     return a / b

ZeroDivisionError: division by zero
```

As you know, answer of 10 / 0 is **Undefined**. So python raise an error names ZeroDivisionError to us.

How to **Handle** it?

Division function

```
>> division(10, '2')  # ?
```

```
Output:
TypeError                                 Traceback (most recent call last)
<ipython-input-16-e15e4b889977> in <module>()
----> 1 division(10, '2')  # ?

<ipython-input-13-d012b66df33b> in division(a, b)
      1 def division(a, b):
----> 2     return a / b

TypeError: unsupported operand type(s) for /: 'int' and 'str'
```

We know python language does Not support strict **Type checking**, So Errors like this are very common during python development.

Now how we Can *handle exceptions* like this ?

Exceptions in Python

1. How does it works?

Use raise to force an exception:

2. Built-in Exception Types in Python

- Exception
- TypeError
- ValueError
- ZeroDivisionError
- KeyError
- ImportError
- AssertionError
- more...

3. How to create custom Exceptions in python?

```
class NormalizerError(Exception):
    pass
```

must Inherit from **Exception** class

```
class PhoneNumError(NormalizerError):
    phone_number: str

    def __init__(self,phone_number, *args):
        super().__init__(*args)
        self.phone_number = phone_number
```

Inherits from **Exception** subclass

4. How to raise Exceptions in python?

```
def normalize_phone(phone_num:str, prefix_num='+98'):
    # Get last 10 digits of phone number (must starts with '9')
    phone_num = phone_num[-10:]
    if len(phone_num) < 10:
        raise PhoneNumError(phone_num, "Phone number Length must be >= 10 .")
    if not phone_num.isnumeric():
        raise PhoneNumError(phone_num, "Phone number is NOT numeric.")
    if not phone_num.startswith('9'):
        raise PhoneNumError(phone_num, "Phone number does NOT start with '9'
.")
    return prefix_num + phone_num   # Also may raises `TypeError`
```

4. How to raise Exceptions in python?

What Exceptions raises in states below?

```
normalize_phone('09379880665')     # OK
normalize_phone('379880665')       # PhoneNumError : Length
normalize_phone('09379880665')     # PhoneNumError : Numeric
normalize_phone('0379880665')      # PhoneNumError : Doesn't start with 9
normalize_phone('9379880665', 0)   # TypeError: str + int
```

Example: User initialization

Maktab Sharif

User initialize Exceptions

Implement User > __init__ method, so that raises UserException when inputs are invalid.

UserException must contains:

- ➔ msg: str -> (Custom message)
- ➔ field: str -> (Field's name with invalid input)
- ➔ data: any -> (The invalid input)

Hints:

1. User id: must be int
2. User name: only contains alphabets and space
3. User phone: must be numeric and starts with '09'
4. User Email: must be ascii and contains only one '@'

```
# User class prototype
class User:
  def __init__(self, id, name, phone, email):
    # TODO : Complete here!
    pass
```

Assertions

Assertions

Assertions are statements that assert or state a fact confidently in your program. Assertions are simply boolean expressions that check if the conditions return true or not. If it is true, the program does nothing and moves to the next line of code. However, if it's false, the program stops and throws an error.

Syntax:

```
assert <condition>
assert <condition>, <error msg>
```

```
def division(a: float, b: float):
    assert b, "Division by Zero"
    return a/b
```

=

```
def division(a: float, b: float):
    if not b:
        raise AssertionError("Division by Zero")
    return a/b
```

Example: User initialization using Assertion

Maktab Sharif

User initialize Exceptions

Implement User > __init__ method, Use assertion to prevent bad inputs.

Hints:

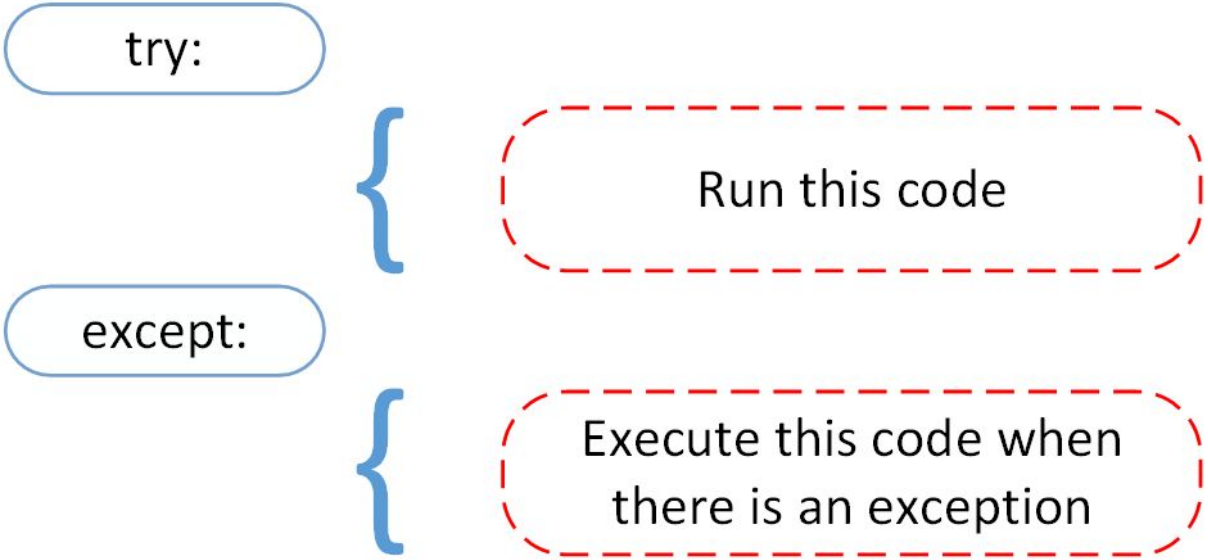
1. User id: must be int
2. User name: only contains alphabets and space
3. User phone: must be numeric and starts with '09'
4. User Email: must be ascii and contains only one '@'

```
# User class prototype
class User:
  def __init__(self, id, name, phone, email):
    # TODO : Complete here!
    pass
```

Exception handling in Python

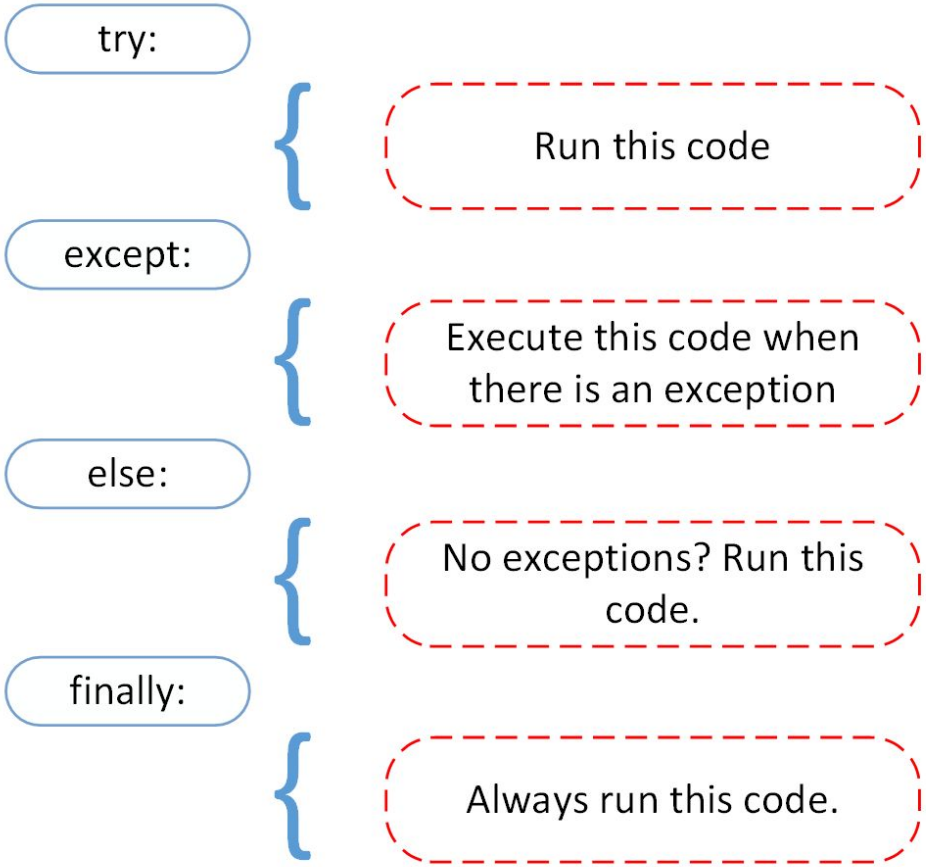
Now how we
Can *handle*
exceptions like this ?

1. How does it works?



2. Keywords

- try
- except
- else (Optional)
- finally (Optional)



```python
a = input('Enter first num: ')
b = input('Enter second num: ')

try:
    res = float(a)/float(b)
except ValueError:
    print('> a and b must be numeric!')
except ZeroDivisionError:
    print('> result is Undefined!')
else:
    print('> Result: ', res)
finally:
    print('== The End ==')
```

OK

```
Enter first num: 12
Enter second num: 23
> Result:  0.5217391304347826
== The End ==
```

ValueError

```
Enter first num: 12.1
Enter second num: 12a
> a and b must be numeric!
== The End ==
```

ZeroDivision Error

```
Enter first num: 10
Enter second num: 0
> result is Undefined!
== The End ==
```

Division exception handling

3. get Exception object with `as` keyword

```
try:
    raise Exception("Hello","World","!")
except Exception as error:
    print(error.args)

Output:
('Hello', 'World', '!')
```

You can simply catch the raised error using as keyword in the except statement.

3. get Exception object with `as` keyword

```
try:
    raise NameError("Hello","World","!")
except Exception as error:
    # It can catch every raised exceptions
(Why?)
    print(error.args)


Output:
('Hello', 'World', '!')
```

You can simply catch the raised error using as keyword in the except statement.

4. Multiple Exceptions Handlings (1 of 2)

```
try:
    ...
except ValueError:
    ...
except AssertionError:
    ...
except KeyError as error:
    ...
```

You can use multiple `except` statement to handling multiple exceptions.

4. Multiple Exceptions Handlings (2 of 2)

Also you can use Tuples for handling multiple exceptions with one `except` statement.

```
try:
    ...
except (ValueError,AssertionError,KeyError):
    ...
except (IndexError,TypeError) as error:
    ...
```

4. Multiple Exceptions Handlings (2 of 2)

Also you can use Tuples for handling multiple exceptions with one `except` statement.

```
try:
    ...
except (ValueError,AssertionError,KeyError):
    ...
except (IndexError,TypeError) as error:
    ...
```

Register user example

```python
class EmailError(NormalizerError):
    pass
```

```python
def normalize_email(email: str):
    """
     Normalize user email using RegEx
    :param email: user email
    :return: normalized and lower email address
    :raises `EmailError`
    """
    import re
    email_regex = r"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"
    if not re.match(email_regex, email):
        raise EmailError("Invalid email")
    return email.lower()
```

first, We define normalize_email function to normalize user email or raise EmailError.

Register user example

Implement register_user function

Now we can create a simple register_user function

```
def register_user(phone,  email, password=None, name=None):
    new_user = {}
    ...
    new_user['phone'] = normalize_phone(phone)
    new_user['email'] = normalize_email(email)
    ...
    return new_user
```

```
phone = input('Enter your phone number: ')
email = input('Enter your email address: ')
```

Getting inputs from user

Final example

Maktab Sharif

Final example

Write a console program with two major functionality:

1. Register user **(name, phone, email:optional , password)**
2. Login user **(username, password)**

Print 'Successfully' message to user
or Print Error and reasons

Hints:

1. Each inputs must be checked and an Exception raised if it's necessary.
2. On the main procedure you should get input from user
3. Use file or pickle to save User informations

```
USER MANAGER PROGRAM
     1. Register
     2. Login
Enter option:
```

```
USER MANAGER > REGISTER
>> phone:
>> password:
>> name:
>> email(Optional):
Registered Successfully!
```

```
USER MANAGER > LOGIN
>> phone:
>> password:
ERROR: Invalid password
```

Advanced topics

- with statement
- Traceback
- Warning Exceptions
- Exception Chaining (raise - from -)