CERTIFIED ASSOCIATE IN PYTHON PROGRAMMING

BY: IMRAN

Exceptions

- Each time your code tries to do something wrong/foolish/irresponsible/crazy/unenforceable, Python does two things:
- it **stops your program**;
- it creates a special kind of data, called an **exception**.
- Both of these activities are called **raising an exception**. We can say that Python always raises an exception (or that an **exception has been raised**) when it has no idea what to do with your code.

Exceptions

- What happens next?
- the raised exception expects somebody or something to notice it and take care of it;
- if nothing happens to take care of the raised exception, the program will be **forcibly terminated**, and you will see an **error message** sent to the console by Python;
- otherwise, if the exception is taken care of and **handled** properly, the suspended program can be resumed and its execution can continue.
- Python provides effective tools that allow you to **observe exceptions, identify them and handle them** efficiently. This is possible due to the fact that all potential exceptions have their unambiguous names, so you can categorize them and react appropriately.

Exceptions

- You know some exception names already. Take a look at the following diagnostic message:
- ValueError: math domain error
- The word highlighted above is just the **exception name**. Let's get familiar with some other exceptions.

Exceptions

- Look at the code in the editor. Run the (obviously incorrect) program.
- value = 1
- value /= 0
- You will see the following message in reply:
- Traceback (most recent call last):
- File "div.py", line 2, in value /= 0
- ZeroDivisionError: division by zero
- This exception error is called `ZeroDivisionError`.

Exceptions

- Look at the code in the editor. What will happen when you run it? Check.
- my_list = []
- x = my_list[0]
- You will see the following message in reply:
- Traceback (most recent call last):
- File "lst.py", line 2, in x = list[0]
- IndexError: list index out of range
- This is the `IndexError`.

Exceptions

- How do you **handle** exceptions? The word try is key to the solution.
- The recipe for success is as follows:
  - first, you have to ***try* to do something**;
  - next, you have to **check whether everything went well**.

- But wouldn't it be better to check all circumstances first and then do something only if it's safe?
- Just like the example

```
first_number = int(input("Enter the first number: "))
second_number = int(input("Enter the second number: "))
if second_number != 0:
    print(first_number / second_number)
else:
    print("This operation cannot be done.")
print("THE END.")
```

Exceptio ns

- Admittedly, this way may seem to be the most natural and understandable, but in reality, this method doesn't make programming any easier. All these checks can make **your code bloated and illegible**.

Exceptions

- ook at the code in the editor. This is the favorite Python approach.
- Note:
  - the try keyword **begins a block of the code** which may or may not be performing correctly;
  - next, Python tries to perform the risky action; if it fails, an exception is raised and Python starts to look for a solution;
  - the except keyword starts a piece of code which will be **executed if anything inside the try block goes wrong** - if an exception is raised inside a previous try block, **it will fail here**, so the code located after the except keyword should provide an **adequate reaction** to the raised exception;
  - returning to the previous nesting level ends the **try-except** section.
- Run the code and test its behavior.

Exceptions

```
first_number = int(input("Enter the first number: "))
second_number = int(input("Enter the second number: "))

try:
    print(first_number / second_number)
except:
    print("This operation cannot be done.")

print("THE END.")
```

Exceptions

```
try:
    print("1")
    x = 1 / 0
    print("2")
except:
    print("Oh dear, something went wrong...")
print("3")
```

Exceptio ns

- This approach has one important disadvantage - if there is a possibility that more than one exception may skip into an except: branch, you may have **trouble figuring out what actually happened**.
- Just like in our code in the editor. Run it and see what happens.
- The message: Oh dear, something went wrong... appearing in the console says nothing about the reason, while there are two possible causes of the exception:
  - non-integer data entered by the user;
  - an integer value equal to 0 assigned to the x variable.

Exceptions

- Technically, there are two ways to solve the issue:
- build two consecutive `try-except` blocks, one for each possible exception reason (easy, but will cause unfavorable code growth)
- use a more advanced variant of the instruction.
- It looks like this:

```
try:
    x = int(input("Enter a number: "))
    y = 1 / x
except:
    print("Oh dear, something went wrong...")
print("THE END.")
```

Exceptions

```
try:
    x = int(input("Enter a number: "))
    y = 1 / x
    print(y)
except ZeroDivisionError:
    print("You cannot divide by zero, sorry.")
except ValueError:
    print("You must enter an integer value.")
except:
    print("Oh dear, something went wrong...")
print("THE END.")
```

Exceptions

- Don't forget that:
  - the except branches are searched in the same order in which they appear in the code;
  - you must not use more than one except branch with a certain exception name;
  - the number of different except branches is arbitrary - the only condition is that if you use try, you must put at least one except (named or not) after it;
  - the except keyword must not be used without a preceding try;
  - if any of the except branches is executed, no other branches will be visited;
  - if none of the specified except branches matches the raised exception, the exception remains unhandled (we'll discuss it soon)
  - if an unnamed except branch exists (one without an exception name), it has to be specified as the last.

Module 2: Strings, Lists, and Exceptions
Part 4: Strings in Action

Exceptio ns

```python
try:
    x = int(input("Enter a number: "))
    y = 1 / x
    print(y)
except ValueError:
    print("You must enter an integer value.")
except:
    print("Oh dear, something went wrong...")
print("THE END.")
```

Exceptions

```
try:
    x = int(input("Enter a number: "))
    y = 1 / x
    print(y)
except ValueError:
    print("You must enter an integer value.")
except:
    print("Oh dear, something went wrong...")
print("THE END.")
```

- Look at the code in the editor. We've modified the previous program - we've removed the ZeroDivisionError branch.
- What happens now if the user enters 0 as an input?
- As there are **no dedicated branches** for division by zero, the raised exception falls into the **general (unnamed) branch**; this means that in this case, the program will say:

Exceptions

- Let's spoil the code once again.
- Look at the program in the editor. This time, we've removed the unnamed branch.
- The user enters 0 once again and:
  - the exception raised won't be handled by ValueError - it has nothing to do with it;
  - as there's no other branch, you should to see this message:

```
try:
    x = int(input("Enter a number: "))
    y = 1 / x
    print(y)
except ValueError:
    print("You must enter an integer value.")
print("THE END.")
```