Introduction to Programming Language (ITP101)

Exceptions

Mulualem Teku

GCIT, Bhutan

Oct 2019

...So Far & Today...

- Computational Thinking
- Core Python objects
  - Functions
  - Lists
  - Tuples
  - Dictionaries

Next...

- Bugs, Debugging
- Exceptions
- Assertions

Brainstorm

1. Programming challenges?

2. What are programming errors? Types?

3. Name some of the error types in Python you have come across.

Brainstorm

1. Programming challenges?
2. What are programming errors? Types?
3. Name some of the error types in Python you have come across.

Brainstorm

1. Programming challenges?

2. What are programming errors? Types?

3. Name some of the error types in Python you have come across.

Introduction

- Bugs
- Debugging
- Exception handling

Errors vs Exceptions

- Errors are everyday ‘friends’ of a programmer.
- Types
  - Parsing (Syntax) errors
  - Logical errors
  - Runtime errors
- Examples?

Exceptions

Events that can modify the control flow through a program.

Why Study?

- Exceptions are inevitable and could be fatal.

  e.g. The Y2K bug, critical systems (e.g. industrial control systems, grid systems)

- Secure Programming (a.k.a. Defensive Programming)
  - Not an option but a necessity now a days.
  - Runtime errors are mostly due to external reasons.

    e.g. Poor user input, malicious input, some sort of failure

- Proper handling of them pays off.

Exceptions

- Are generated automatically on errors.
- Built-in vs User-defined exceptions
- Can be triggered and handled by our code.
- Many languages offer exception-handling constructs.

Example

Python, Java, C++, Eiffel, Modula-3

Exception Handling

Generally, a two-phase process:

1. Detection of exception condition
    - Interpreter *raises* (throws/triggers/generates) the exception.
    - Programmer too can raise explicitly.
2. Exception handling

   e.g. Ignore error, log error, abort program, remedial actions, etc

Exceptions (Python)

- Some standard/built-in exceptions you've probably encountered:
  - *NameError* - access uninitialized variable
  - *SyntaxError*
  - *ZeroDivisionError*
  - *KeyError* - access non-existing dictionary key
  - *IndexError* - access out-of-range index
  - *IOError* - input/output (e.g. in file read/write)
  - *TypeError* - operations with invalid type.
  - *ImportError*
- On error, the *default exception handler* throws the error messages + stack trace.

The Constructs

- Exceptions can be detected by a `try` statement.
- Flavors :
  - `try...except...[else]`
  - `try...finally`

try...except

```
try:
   <statements>                        # suspicious code
except <e1>:
    <statements>                       # if <e1> was raised
except (e2, e3, ...eN):
    <statements>                       # if any of e2...eN was raised
except:
      <statements>                     # for all other exceptions
else:                                  # optional else block
    <statements>
```

Example

```
try:
     a , b = 8, 0
     c = a / b
except ZeroDivisionError:
     print ( "Oops! Check your numbers." )
```

Example

```
try:
    n ** 4
    "2" + 2
except(NameError, TypeError):
    print ("Some exception occured!")
else:
    print ("No exception occured!")
```

try...finally

```
try:
   <statements>
finally:
   <statements>               # Always run this code
```

- Unlike an `except` clause, `finally` is not used to handle exception.
- The clause executes regardless of exceptions within the `try` clause.
- Useful to specify cleanup actions that must occur, regardless of exception.

  e.g. File close, server disconnects, etc

Example

```
try:
     n = float(input("Enter your number:"))
     d = 2 * n

finally:
     print ("Who can stop me from executing?")

print ("Double = ", d )
```

Exception Arguments

- Exceptions can have arguments.
- Are values that give additional info about the error (if any), usually error string, number and location.
- Captured by supplying a variable in the `except` clause.

exception args

```
try:
   try_block
except <single or multiple exception>, argument:
   exception handler
```

- An alternative is by accessing the `exc_info()` method of the `sys` module (returns a 3-tuple info).

Raising Exceptions

- To explicitly raise exceptions, use the `raise` statement.

The raise statement

```
raise <exception_to_be_raised> [, args]
```

- If no exception supplied with the `raise` statement, the last exception (if any) in the current try block is re-raised; otherwise, `TypeError` (no exception to re-raise).

Example

```
try:
     raise NameError
except NameError:
     print ‘Exception ocurred!’
     raise
```

Assertions

- Are diagnostic predicates which *must evaluate to true*.
- If false, an `AssertionError` exception is thrown.
- Think of them as conditional raise i.e. `raise-if/raise-if-not`

Syntax:

```
assert <test>
```

Example

```
>>>assert 2 == ‘2’

>>>assert range(2) == [1,2]



>>>def f(n):
       assert n>0            # must be positive
       return math.sqrt(n)
```