

Introduction to Python:

Day 3 - Exception Handling

Ryden Butler

Syntax Errors

- Errors related to language structures
- Forgotten symbols, typos, or confusing object names
- Pre-runtime; parser doesn't understand; fatal
- Check the \wedge

```
In [35]: if True:
...:     print("hello")
...:     else
...:     print("bye")
```

```
File "<ipython-input-35-c43b9c02df68>", line 3
    else
    ^
```

SyntaxError: invalid syntax

Runtime Errors

- Errors during the execution of program
- e.g. TypeError, NameError, ZeroDivisionError

```
>>> callMe = "Maybe"
```

```
>>> print(callme)
```

```
Traceback (most recent call last):
```

```
  In line 2 of the code you submitted:
```

```
    print(callme)
```

```
NameError: name 'callme' is not defined
```

```
>>> print("you cannot add text and numbers" + 12)
```

```
Traceback (most recent call last):
```

```
  In line 1 of the code you submitted:
```

```
    print("you cannot add text and numbers" + 12)
```

```
TypeError: Can't convert 'int' object to str
```

Semantic Errors

- The program will run successfully but the output is not what you expected
- Task: create a program that computes the average of two numbers $\left(\frac{x+y}{2}\right)$

```
>>> x = 3
>>> y = 4
>>> average = x + y / 2
>>> print(average)
```

5.0

- What happened?
- Very common and can be difficult to diagnose

Reviewing Errors

- A *syntax error* happens when python can't understand your code
`x *= 2`
- A *runtime error* happens when python understands your code, but runs into trouble following your instructions
`12 / 0`
- A *semantic error* happens when python understands and executes your code, but doesn't behave as expected
do you mean `x = y` or `x == y` ?

Debugging Tips

Double-check the following:

- Objects do not share names with reserved / keywords

```
>>> import keyword
>>> keyword.kwlist
```
- A colon is included after `for`, `while`, `if`, `else`, `def` etc.
- Parentheses and quotation marks are closed
- `=` and `==` have not been confused
- Code is indented correctly (spaces in empty lines count)
- Indexing begins at 0

Exceptions

- Used when we expect errors to (re-)occur
- Write code that executes only for specific error(s)
- Multiple errors can be handled separately

```
try:  
... # tries to execute some code  
except TypeError:  
... # runs if any Type Error is raised  
except AttributeError:  
... # runs for any Attribute Error  
else:  
... # runs if no exception is raised  
finally:  
... # always runs
```

More Exceptions

Double-check the following:

- You can create custom exceptions using what we've learned about classes
- These will be helpful in targeting particular types of errors
- We will cover many examples today
- These are most useful in:
 - web scraping
 - package development

What is Unit Testing?

- Write tests before / alongside your code
- Tests the smallest possible unit of your code
- Automate tests
- Test-driven development

Why Unit Test?

- Find bugs quickly (especially semantic errors)
- Forces code structure
- Allows integration of multiple functions
- Makes it easy to return to old code
- Make changes to code without violating dependencies
- Better to incorporate into your workflow

Example Test

```
import unittest # unit testing module
import script # the script you want to test

class mytest(unittest.TestCase):
    def test_one(self):
        self.assertEqual("result", script.func(input))

    def test_two(self):
        thing1 = script.func(input1)
        thing2 = script.func(input2)
        self.assertNotEqual(thing1, thing2)

# if you want to run the test with this script
if __name__ == '__main__':
    unittest.main()
```

Functions

- `self.assertEqual(,)`
- `self.assertNotEqual(,)`
- `self.assertTrue(,)`
- `self.assertFalse(,)`
- `self.assertRaises(,)`
- More at: <https://docs.python.org/3/library/unittest.html>

Example Test

```
import unittest

class TestStringMethods(unittest.TestCase):
    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')
    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())
    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails
        with self.assertRaises(TypeError):
            s.split(2)
```

Functions

- Allow tighter control over loops
- `break`: stops a loop
- `continue`: advances to next iteration
- `else`: executed if no `break` occurs

```
for n in range(2, 10):  
    for x in range(2, n):  
        if n % x == 0:  
            print(n, 'equals'), x, '*', n // x  
            break  
    else:  
        print(n, 'is a prime number')
```

Poster Info

- Templates (Overleaf)
- Guidance