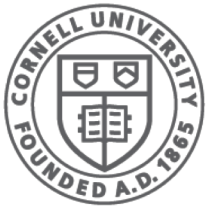
http://www.cs.cornell.edu/courses/cs1110/2019sp

Lecture 11: Asserts & Error Handling

CS 1110

Introduction to Computing Using Python



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Postal Function (v1)

```
def print_mailing_label(num, st, city, state, zip):
    """
    prints out address in standard mailing format
    """
    print("Ship to:")
    print(str(num) + " " + st)
    print(city+", "+state+"  "+zip)
```

postal_v1.py

Fine as long as nothing goes wrong….

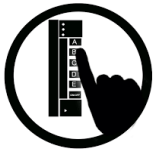
```
>>> import postal_v1
>>> postal_v1.print_mailing_label(100, "Main Street", "Ithaca", "NY", "14850")
Ship to:
100 Main Street
Ithaca, NY  14850
>>>
```

Postal Function (v1) with Error

```
def print_mailing_label(num, st, city, state, zip):
    """
    prints out address in standard mailing format
    """
    print("Ship to:")
    print(str(num) + " " + st)
    print(city+", "+state+"  "+zip)
```

postal_v1.py

```
>>> postal_v1.print_mailing_label(100, "Main Street", "Ithaca", "NY", 14850)
Ship to:
100 Main Street
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "11-asserts_errors/postal_v1.py", line 14, in print_mailing_label
    print(city+", "+state+"  "+zip)
TypeError: must be str, not int
```



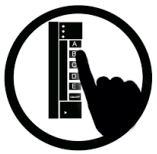
Where is the errror?

```
def print_mailing_label(num, st, city, state, zip):
    """
    prints out address in standard mailing format
    """
    print("Ship to:")
    print(str(num) + " " + st)
    print(city+", "+state+"  "+zip)
```

postal_v1.py

A: the function call
B: the concatenation
C: the specification
D: A & B
E: A & B & C

```
>>> postal_v1.print_mailing_label(100, "Main Street", "Ithaca", "NY", 14850)
Ship to:
100 Main Street
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "11-asserts_errors/postal_v1.py", line 14, in print_mailing_label
    print(city+", "+state+"  "+zip)
TypeError: must be str, not int
```



Postal Function (v2)

```
def print_mailing_label(num, st, city, state, zip):
    """
    prints out address in standard mailing format
    Preconditions
    num: an integer with 4 or fewer digits
    st: str representing the street name
    city: str the city name
    state: a 2-digit all-caps str representing the state
    zip: a 5 digit string
    """
    print(str(num) + " " + st)
    print(city+", "+state+"  "+zip)
```

postal_v2.py

A: the function call
B: the concatenation
C: the specification
D: A & B
E: A & B & C

Can we be more helpful?

```
>>> postal_v2.print_mailing_label(100, "Main Street", "Ithaca", "NY", 14850)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "11-asserts_errors/postal_v2.py", line 21, in print_mailing_label
    print(city+", "+state+"  "+zip)
TypeError: must be str, not int
```

Assert Statements

```
assert <boolean>                # Creates error if <boolean> false
assert <boolean>, <string>      # As above, but displays <String>
```

- A way to force an error
  - Why would you do this?
- Enforce preconditions!
  - Put precondition as assert.
  - If violate precondition, the program crashes
- Provided code in A3 uses asserts heavily
- Will do yourself in A4

Postal Function (v3)

```
def print_mailing_label(num, st, city, state, zip):
    assert type(num) == int, "street number must be an int"
    assert type(st) == str, "street name must be a str"
    assert type(city) == str, "city must be a str"
    assert type(state) == str, "state must be a str"
    assert len(state) == 2, "state must be 2-digits"
    assert type(zip) == str, "zip code must be a str"
    print("Ship to:")
    print(str(num) + " " + st)
    print(city+", "+state+"  "+zip)
```

postal_v3.py

```
>>> postal_v3.print_mailing_label(100, "Main Street", "Ithaca", "NY", 14850)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "11-asserts_errors/postal_v3.py", line 24, in print_mailing_label
    assert type(zip) == str, "zip code must be a str"
AssertionError: zip code must be a str
```

← *much better!*

Enforcing Preconditions is Tricky!

Want the state abbreviation to be:

- A string
- 2 digits
- An actual US state

→ Use a helper function to enforce preconditions!

Postal Function (v4)

```
def good_state(state):
    return type(state) == str and len(state) == 2 and state == "NY" # etc.


def print_mailing_label(num, st, city, state, zip):
    . . .
    assert good_state(state), "state is ill-formatted"
    print("Ship to:")
    print(str(num) + " " + st)
    print(city+", "+state+"  "+zip)
```

postal_v4.py

```
>>> postal_v4.print_mailing_label(100, "Main Street", "Ithaca", "NX", "14850")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "11-asserts_errors/postal_v4.py", line 25, in print_mailing_label
    assert good_state(state), "state is ill-formatted"
AssertionError: state is ill-formatted
```

What if we want lots of postal functions?

`print_mailing_label`

`print_european_mailing_label`

`make_911_compliant`

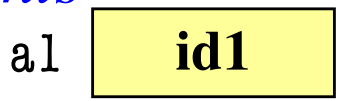
Do all of these functions have to check the same preconditions?

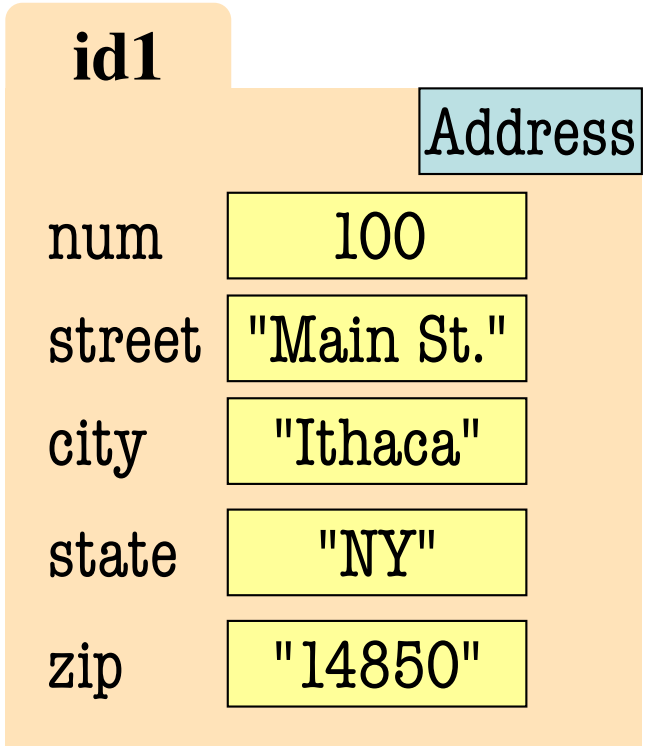
Redundancy Alert!

What we really want is an Address Object

Class Attributes can have *invariants*

- Limit the attribute values
- Example: zip is a 5-digit str
- Get an error if you violate
- Now when you use an object of that class, you can rely on the invariants being true






```
>>> import postal_v5
>>> a1 = postal_v5.Address(100, "Main Street", "Ithaca", "NY", "14850")
>>> postal_v5.print_european_mailing_label(a1)
```

An:

Main Street 100

14850 Ithaca

Make an object with bad inputs and you'll get an error.

Postal Function (v5)

```
def print_european_mailing_label(addr):
    """
    prints out the address in european address format
    Precondition: addr is of type Address
    """
    print("An:")
    print(addr.st+ " " +str(addr.num))
    print(addr.zip+" "+addr.city)
```

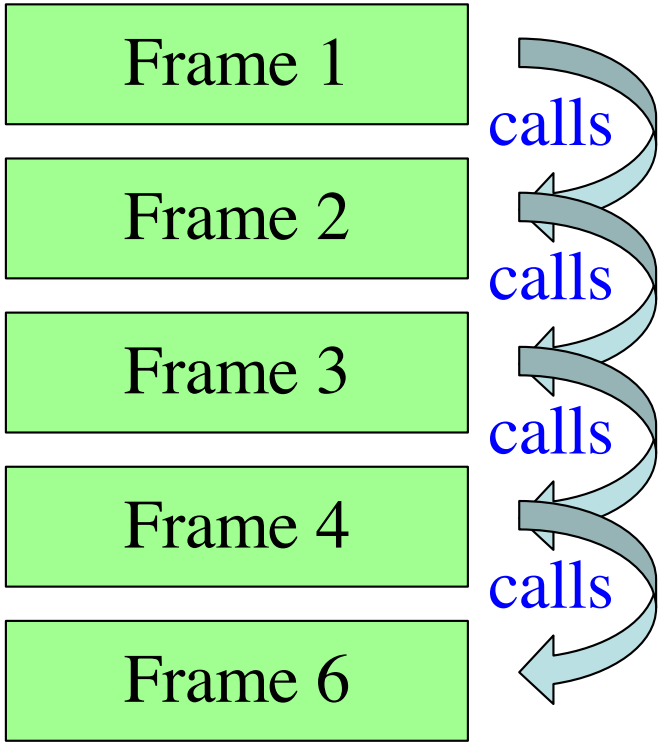
postal_v5.py

```
>>> import postal_v5
>>> a1 = postal_v5.Address(100, "Main Street", "Ithaca", "NY", "14850")
>>> postal_v5. print_european_mailing_label(a1)
Ship to:
Main Street 100
14850 Ithaca
```

Let's draw this out on the board!

Recall: The Call Stack

- Functions are "stacked"
  - Cannot remove one above w/o removing one below
  - Sometimes draw bottom up (better fits the metaphor)
- Stack represents memory as a "high water mark"
  - Must have enough to keep the **entire stack** in memory
  - Error if cannot hold stack



Errors and the Call Stack

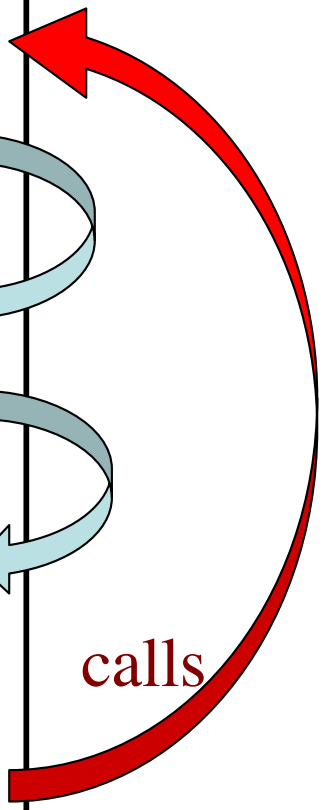
```
# error.py


def function_1(x,y):
    return function_2(x,y)

def function_2(x,y):
    return function_3(x,y)

def function_3(x,y):
    return x/y # crash here


function_1(1,0)
```



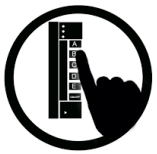
Errors and the Call Stack

```
1   # error.py
2
3   def function_1(x,y):
4       return function_2(x,y)
5
6   def function_2(x,y):
7       return function_3(x,y)
8
9   def function_3(x,y):
10      return x/y # crash here
11
12  function_1(1,0)
```

Crash produces the call stack:

```
Traceback (most recent call last):
  File "error.py", line 12, in <module>
    function_1(1,0)
  File "error.py", line 4, in function_1
    return function_2(x,y)
  File "error.py", line 7, in function_2
    return function_3(x,y)
  File "error.py", line 10, in function_3
    return x/y # crash here
ZeroDivisionError: division by zero
```

Make sure you can see line numbers in Atom.



Question: What line has the error?

(assume there are clear preconditions)

```
1   # error.py
2
3   def function_1(x,y):
4       return function_2(x,y)
5
6   def function_2(x,y):
7       return function_3(x,y)
8
9   def function_3(x,y):
10      return x/y # crash here
11
12  function_1(1,0)
```

Crash produces the call stack:

```
Traceback (most recent call last):
  File "error.py", line 12, in <module>
    function_1(1,0)
  File "error.py", line 4, in function_1
    return function_2(x,y)
  File "error.py", line 7, in function_2
    return function_3(x,y)
  File "error.py", line 10, in function_3
    return x/y # crash here
ZeroDivisionError: division by zero
```

A: 12 B: 4 C: 7 D: 10 E: 10 & 12

Update: unless you know what the preconditions are, this is not a well-specified question. Sorry!