

Using Color Objects in A3

- New classes in intros
 - RGB, CMYK, and HSV
- Each has its own attributes
 - RGB:** red, blue, green
 - CMYK:** cyan, magenta, yellow, black
 - HSV:** hue, saturation, value
- Attributes have *invariants*
 - Limits the attribute values
 - Example: red is int in 0..255
 - Get an error if you violate

```

c = id1
r = 128
                
```

id1	
RGB	
red	128
green	0
blue	0

```

>>> import intros
>>> c = intros.RGB(128,0,0)
>>> r = c.red
>>> c.red = 500 # out of range
AssertionError: 500 outside [0,255]
                
```

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
if __name__ == '__main__':
    print(function_1(1,0))
                
```

Crashes produce the call stack:

```

Traceback (most recent call last):
  File "error.py", line 20, in <module>
    print(function_1(1,0))
  File "error.py", line 8, in function_1
    return function_2(x,y)
  File "error.py", line 12, in function_2
    return function_3(x,y)
  File "error.py", line 16, in function_3
    return x/y
                
```

10/3/19 Asserts & Error Handling 2

Determining Responsibility

```

def function_1(x,y):
    """Returns: result of function_2
    Precondition: x, y numbers"""
    return function_2(x,y)
def function_2(x,y):
    """Returns: x divided by y
    Precondition: x, y numbers"""
    return x/y
print(function_1(1,0))
                
```

Traceback (most recent call last):

```

File "error1.py", line 32, in <module>
    print(function_1(1,0))
File "error1.py", line 18, in function_1
    return function_2(x,y)
File "error1.py", line 28, in function_2
    return x/y
ZeroDivisionError:
                
```

Where is the error?

Assert Statements

- Form 1:** `assert <boolean>`
 - Does nothing if boolean is True
 - Creates an error if boolean is False
- Form 2:** `assert <boolean>, <string>`
 - Very much like form 2
 - But error message includes the string
- Statement to **verify a fact is true**
 - Similar to `assert_equals` used in unit tests
 - But more versatile with complete **stack trace**

Example: Anglicizing an Integer

```

def anglicize(n):
    """Returns: the anglicization of int n.
    Precondition: n an int, 0 < n < 1,000,000"""
    assert type(n) == int, repr(n)+' is not an int'
    assert 0 < n and n < 1000000, repr(n)+' is out of range'
    # Implement method here...
                
```

Check (part of) the precondition

Error message when violated

Enforcing Preconditions is Tricky!

```

def lookup_netid(nid):
    """Returns: name of student with netid nid.
    Precondition: nid is a string, which consists of
    2 or 3 letters and a number"""
    assert type(nid) == str, repr(nid) + ' is not a string'
    assert nid.isalnum(), repr(nid)+' is not letters/digits'
                
```

Returns True if s contains only letters, numbers.

Does this catch all violations?

Using Function to Enforce Preconditions

```
def exchange(curr_from, curr_to, amt_from):
    """Returns: amount of curr_to received.
    Precondition: curr_from is a valid currency code
    Precondition: curr_to is a valid currency code
    Precondition: amt_from is a float"""
    assert ?????, repr(curr_from) + ' not valid'
    assert ?????, repr(curr_to) + ' not valid'
    assert type(amt_from)==float, repr(amt_from)+' not a float'
```

Using Try-Except

```
try:
    result = input('Number: ') # g
    x = float(result) # convert to float
    print('The next number is '+str(x+1))
except:
    print('That is not a number!')
```

Conversion may crash!

Execute if crashes

- Similar to if-else
- But always does the try block
 - Might not do **all** of the try block

Try-Except is Very Versatile

```
def isfloat(s):
    """Returns: True if string
    s represents a float"""
    try:
        x = float(s)
        return True
    except:
        return False
```

Conversion to a float might fail

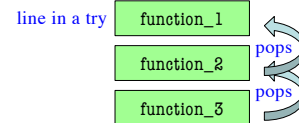
If attempt succeeds, string s is a float

Otherwise, it is not

Try-Except and the Call Stack

```
# recover.py
def function_1(x,y):
    try:
        return function_2(x,y)
    except:
        return float('inf')
def function_2(x,y):
    return function_3(x,y)
def function_3(x,y):
    return x/y # crash here
```

- Error “pops” frames off stack
 - Starts from the stack bottom
 - Continues until it sees that current line is in a try-block
 - Jumps to except, and then proceeds as if no error



Tracing Control Flow

```
def first(x):
    print('Starting first.')
    try:
        second(x)
    except:
        print('Caught at first')
    print('Ending first')
```

```
def second(x):
    print('Starting second.')
    try:
        third(x)
    except:
        print('Caught at second')
    print('Ending second')
```

```
def third(x):
    print('Starting third.')
    assert x < 1
    print('Ending third.')
```

What is the output of first(2)?

Tracing Control Flow

```
def first(x):
    print('Starting first.')
    try:
        second(x)
    except:
        print('Caught at first')
    print('Ending first')
```

```
def second(x):
    print('Starting second.')
    try:
        third(x)
    except:
        print('Caught at second')
    print('Ending second')
```

```
def third(x):
    print('Starting third.')
    assert x < 1
    print('Ending third.')
```

What is the output of first(0)?