

Using Color Objects in A3

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

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

1

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

Make sure you can see line numbers in Pulsar.

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
```

ZeroDivision

Where is the error?

3

This Makes Debugging Tricky

- Place error is detected != place error occurred
 - It can be further up in the call stack
 - It can even happen earlier in the code
 - Error messages are not enough to debug
- Debugging requires “stepping-through” code
 - You execute each line of code in your mind
 - You determine what Python does at that step
 - You verify that the two match (if not, bug)

4

Debugging Requires Visualization

Can Use Special Tools

```
Global
last_name_first: str

Frames
last_name_first
n: Walker White
end_first: 6
first: Walker
last: White
Return value
White, Walker
```

Can Use print

```
>>> last_name_first('Walker White')
Start last_name_first(Walker White)
end_first is "6"
first is "Walker"
last is "White"
result is "White, Walker"
Finish last_name_first(Walker White)

'White, Walker'
```

5

Case Study: Romanize

- Roman Numerals use letters instead of numbers
 - I = 1, V = 5, X = 10, L = 50, and C = 100
 - Can go higher, but will stop there for now
- Create other numbers by concatenation
 - II = 2, III = 3
 - LV = 55, LVIII = 53
- If a smaller number comes first, we subtract
 - IV = 4, IX = 9
 - VL = 40, XCVL = 94

6

Dynamic Bug Search

- Start with a small number of print statement
 - Put them in the area you think error is
 - Leave large sections of code between prints
- Find the first print statement that is “wrong”
 - Error is must be before it (and after last print?)
 - Add more print statements in this area
 - Repeat this process as needed
- Problem is when cannot find anything wrong
 - Pick an area that was hard and add more prints

7

Aiding the Search Process

- In practice, searching is hard
 - Errors can go a long time until we catch them
 - Errors often seen by user, not developer
- Would be easier if errors just *crashed*
 - A crash always gives a stack trace
 - The user can just email that to you
- We do this with **assert statements**
 - Crash the program when an error is seen

9/29/26

Asserts & Debugging

8

8

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 1
 - 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**

9

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

10

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?

11

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'
```

12