Lecture 4a
(I) Types of Error

Humans make mistakes
You are only human
Therefore, you will
make mistakes ☺

Debugging

- Means to remove errors (“bugs”) from a program.
- After debugging, the program is not necessarily error-free.
  - It just means that whatever errors remain are harder to find.
  - This is especially true for large applications.

Common Types of Errors

- Omitting return statement

```
def square(x):
   x * x
```

- Incompatible types

```
def square(x):
   return x * x

>>> print('Answer is : ' + square(5)
```

- Incorrect no. of arguments

```
>>> square(3,5)
```

Common Types of Errors

- Syntax error

```
def double(x):
   return 2x
```

- Arithmetic error

```
x = 3
y = 0
x/y
```

- Undeclared variables

```
>>> x = 2
>>> x + k
```

Common Types of Errors

- Infinite loop

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n-1)

fact(2.1)
fact(-1)

def fact_iter(n):
  counter, result = n, 1
  while counter!=0:
    counter, result = counter-1, result*counter
  return result

fact_iter(-1)
```

To cover after LT 5 and 6

Common Types of Errors

- Numerical imprecision (floating point error)

```
>>> ((10)**(1/2))**2

10.000000000000002
```

Common Types of Errors

- Logic

```
def check(x):
  if x > 100:
    return “Big.”
  elif x > 2000:
    return “Very Big!”
```

How to debug?

- Think like a detective
  - Look at the clues: error messages, variable values.
  - Eliminate the impossible.
  - Run the program again with different inputs.
  - Does the same error occur again?

Print Statements

```
x = 5
y = 10
z = 15
def f(x):
    print('1. ', x, y, z)
    def g(y):
        print(‘2. ', x, y, z)
        def h(z):
            print(‘3. ', x, y, z)
            return x + y + z
        return h(y)
    return g(x)


>>> f(6)
```

Summary

- Debugging often takes up more time than coding
- More an art than a science
- Play detective!
- Do it systematically
- Avoid debugging with good programming practices

Lecture 4a
(II) Test Cases

Importance of Test Cases

- Ensure that it performs its functions as intended

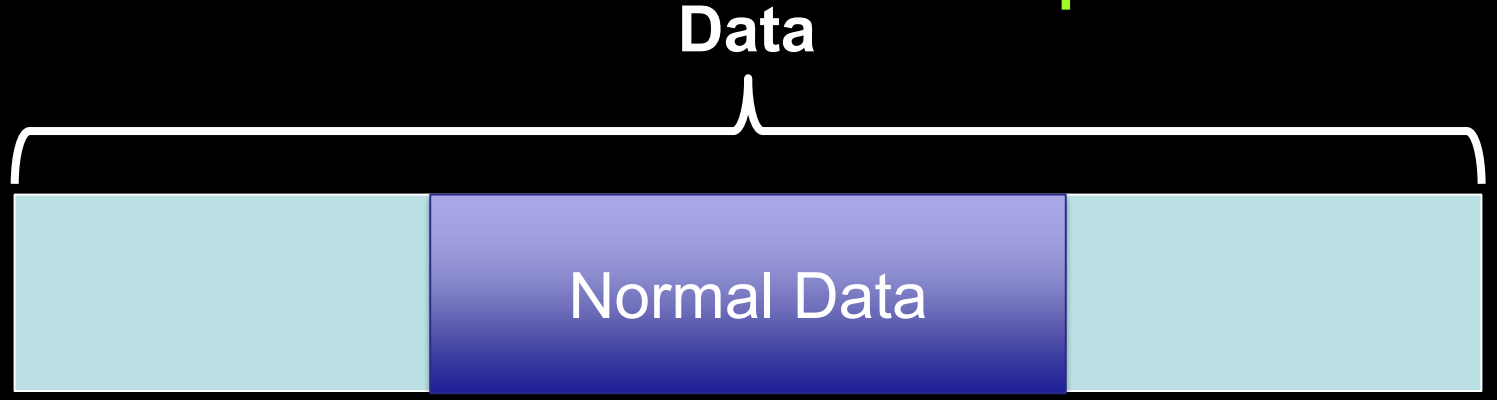
Appropriate Test Cases

Consist of

- Normal Data Values
- Extreme/Boundary Data Values
- Abnormal Data Values
- Volume Data Values

Normal Data Values

- Data that will normally be entered into system
- System should accept and
- Output should be checked to ensure that it is the same as expected



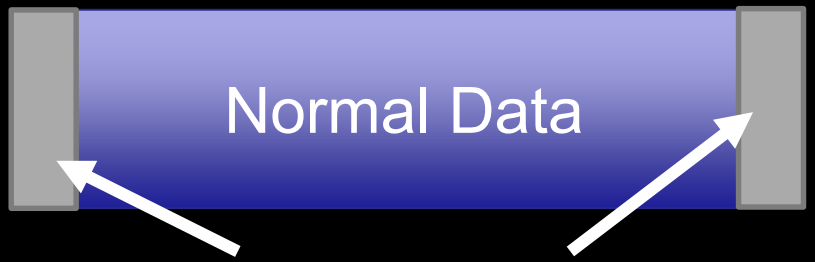
Normal Data Values

```
def percentage(score,total):
    return (score/total)*100
```

```
print(percentage(20,80)==25.0)
print(percentage(40,80)==50.0)
```

Extreme / Boundary Data Values

- Normal data values that are the absolute limits of the normal range
- Helps ensure that all normal values are accepted and processed correctly



Extreme / Boundary Data Values

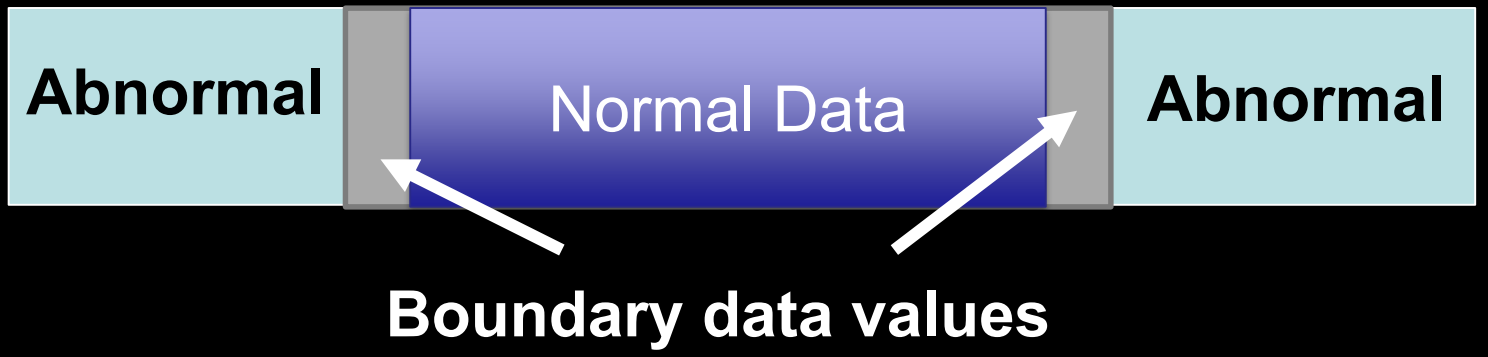
```
def percentage(score,total):
    return (score/total)*100



print(percentage(0,80)==0.0)
print(percentage(60,60)==100.0)
```

Abnormal Data Values

- Should not be accepted by the system
  - Invalid values
- Ensures that invalid values do not break the system



Abnormal Data Values

```
def percentage(score,total):
    return (score/total)*100



print(percentage(-10,80)==-12.5)
print(percentage(120,60)==200.0)
```

Abnormal Data Values

```
def percentage(score,total):
    if score<0 or score>total:
        return ‘Error‘
    else:
        return (score/total)*100
```

```
print(percentage(-10,80)=='Error')
print(percentage(120,60)=='Error')
```

Volume Data Values

- For programs which reads large amount of data
- Input multiple/large data
- Tests if the program is efficient (has reasonable response time)

Summary

- Using appropriate test cases is important to ensure that the program performs as intended
- Consist of:
  - Normal data values
  - Extreme/Boundary data values
  - Abnormal data values
  - Volume data values