10. Logical Maneuvers

Topics:
- Boolean Variables
- Boolean Functions
- Exceptions
- Assertions
- Type Checking
- Try-Except
Boolean Variables
It is possible to assign a float value to a variable:

\[ a = 1.3 \]
\[ b = 10.1 \]
\[ c = 3.7 \]
\[ r = -b + \text{math.sqrt}(b*b-4*a*c))/(2*a) \]
It is possible to assign a string value to a variable:

\[
\begin{align*}
m &= '7' \\
d &= '4' \\
y &= '1776' \\
\text{date} &= m + '/'} + d + '/'} + y
\end{align*}
\]
Review: Variables and Booleans

It is possible to assign a boolean value to a variable:

L = 1
R = 2
x = 1.3
inside = (L<=x) and (x<=R)
Boolean Variables

As the course progresses you will be dealing with logical situations that are increasingly complicated.

Boolean variables are a handy way of keeping track of what is going on.
Example: Leap Year

Gregorian Calendar Rule:

Y is a leap year if it is a century year that is divisible by 400 or a non-century year that is divisible by 4.

Leap years: 1904, 2000, 2016

Not leap years: 1900, 2015
Example: Leap Year

Gregorian Calendar Rule:

Y is a leap year if it is a century year that is divisible by 400 or a non-century year that is divisible by 4.

centuryYear = (Y % 100 == 0)
if centuryYear:
    LY = (Y % 400 == 0)
else:
    LY = (Y % 4 == 0)

Y is a positive int.
LY is assigned the value True if Y is a leap year and False otherwise.
Boolean Functions
Boolean Functions

A function can return a boolean value.

This can be a handy way of encapsulating a complicated computation that culminates in the production of a True value or a False value.
Example: Intersecting Squares

Given two unit squares and a point, when is the point inside both squares?

A unit square has side length one.
Point in a Unit Square

Must have:

\[ a \leq x \leq a+1 \]
\[ b \leq y \leq b+1 \]

\[ x_{OK} = (a \leq x \leq a+1) \]
\[ y_{OK} = (b \leq y \leq b+1) \]
Point in a Unit Square

```python
def inS(a, b, x, y):
    """ Returns True if (x,y) is inside the square with vertices (a,b), (a+1,b), (a,b+1), and (a+1,b+1). Otherwise, returns False."""
    xOK = (a<=x<=a+1)
    yOK = (b<=y<=b+1)
    z = (xOK and yOK)
    return z
```
Using \textit{inS}

\[ z_2 = \text{inS}(a_1,b_1,x,y) \text{ and inS}(a_2,b_2,x,y) \]

\( z_2 \) is True if and only if \((x,y)\) is inside

(i) the unit square with lower left vertex \((a_1,b_1)\).

and also

(ii) the unit square with lower left vertex \((a_2,b_2)\).
Exceptions

Exceptions are errors that occur while your program is running. The program stops running when an exception is “raised.”

There are many types of exceptions.

Here are some examples...
ValueError

```python
>>> t = int('12F')
ValueError: invalid literal for int() with base 10: '123F'
```

In English:

The `int` function does not accept a string unless it encodes a number.
>>> from superMath import sqrt

ImportError: No module named superMath

In English:
You cannot import stuff from a nonexistent module or a module that is not in the same working directory
In English:

```python
>>> from math import SquareRoot
ImportError: cannot import name SquareRoot
```

the math module does not contain a function named SquareRoot
>>> x = 3
>>> x = y+2
NameError: name 'y' is not defined

In English:
The variable y does not exist.
TypeError

```python
>>> x = 3
>>> s = 'abc'
>>> t = s/x
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

In English:

You cannot divide a string by a number.
The square root function requires a number.
ZeroDivisionError

>>> x = 3.0/0.0
ZeroDivisionError: float division by zero

In English:
Cannot divide by zero.
Assertions

They enable you to generate exceptions if something is wrong.

A good way to check that your code is doing what it should be doing.

A good way to focus on pre- and post- conditions during the program development phase.
Assertions: How They Work

Syntax:

```
assert B, S
```

B is a boolean expression.

S is a string.

If B is not true, then string S is printed and an exception is “raised”.

Otherwise, nothing is done.
Checking Pre-, Post- Conditions

Typical:

1. At the start of a function body, are the preconditions satisfied?
2. At the end of the function body, does the value returned have the required properties?
Checking Pre-, Post Conditions

def sqrt(x):
    """ Returns an approximate square root of x in that |L*L-x| <= .001
    """

PreC: x is a positive number. """
def sqrt(x):

    assert x>0, 'The sqrt function requires a positive argument.'

    L = float(x)
    L = (L+x/L)/2
    L = (L+x/L)/2
    L = (L+x/L)/2
    L = (L+x/L)/2

    assert abs(L*L-x)<=.001,
    'Inaccurate Square Root'

    return L
Type Checking

Use `assert` and the function `isinstance`
How `isinstance` Works

It is a boolean-valued function with two arguments.

- `isinstance(x, int)`
  - True if variable x houses an int value
  - Otherwise, False

- `isinstance(x, float)`
  - True if variable x houses a float value
  - Otherwise, False

- `isinstance(x, str)`
  - True if variable x houses a string value
  - Otherwise, False
Using `isinstance`

Guard against the user passing a string to `sqrt`:

def sqrt(x):
    assert isinstance(x,float) or isinstance(x,int),
    print ‘x must be type int or float’
:
The Try–except Construction

A graceful way to handle exceptions
Example: Try-Except

```
try:
    from AintNoMath import sqrt
    print 'AintNoMath.sqrt unavailable'

except ImportError:
    from math import sqrt
    print 'AintNoMath.sqrt is not available'

# Code that uses sqrt...
a = 9; x = sqrt(a); print a,x
```

If the green code triggers an ImportError exception, then the mauve code is executed and “sqrt” comes from the math module. Otherwise sqrt comes from AintNoMath
Try-Except Construction

try:

Code that may generate a particular exception

except Name of Exception :

Code to execute if the particular exception is found