Introduction to Programming in Python

Booleans and Conditionals

Dr. Bill Young
Department of Computer Science
University of Texas at Austin

Last updated: June 4, 2021 at 11:04

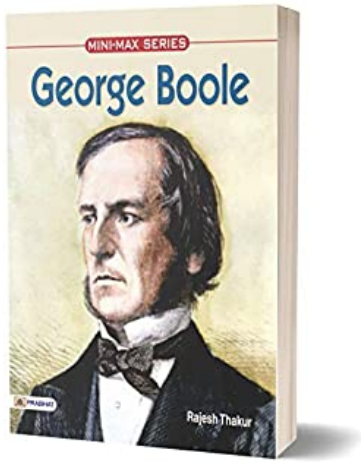
Booleans

So far we've been considering *straight line code*, meaning to do one statement after another.

But often in programming, you need to ask a question, and *do different things* based on the answer.

Boolean values are a useful way to refer to the answer to a yes/no question.

The Boolean **constants** are the values: True, False. A Boolean **expression** evaluates to a Boolean value.



Using Booleans

```
>>> import math
>>> b = ( 30.0 < math.sqrt( 1024 ))
>>> print( b )
True
>>> x = 1              # statement
>>> x < 0              # boolean expression
False
>>> x >= -2            # boolean expression
True
>>> b = ( x == 0 ) # statement containing
                     # boolean expression
>>> print (b)
False
```


Boolean Context

In a **Boolean context**—one that expects a Boolean value—False, 0, "" (the empty string), and None all stand for False and *any other value* stands for True.

```
>>> bool("xyz")
True
>>> bool(0.0)
False
>>> bool("")
False
>>> if 4: print("xyz")      # boolean context
xyz
>>>if 4.2: print("xyz")
xyz
>>> if "ab": print("xyz")
xyz
```

This is very useful in many programming situations.

Comparison Operators

The following comparison operators are useful for comparing numeric values (or strings):

Operator	Meaning	Example
<	Less than	x < 0
<=	Less than or equal	x <= 0
>	Greater than	x > 0
>=	Greater than or equal	x >= 0
==	Equal to	x == 0
!=	Not equal to	x != 0

Each of these returns a Boolean value, True or False.

```
>>> import math
>>> x = 10
>>> ( x == math.sqrt( 100 ))
True
```

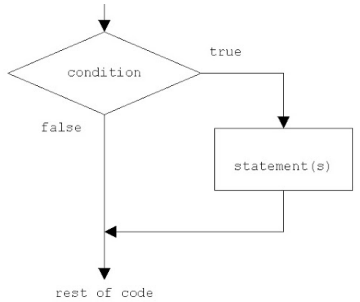

One Way If Statements

It's often useful to be able to perform an action *only if* some conditions is true.

General form:

```
if boolean-expression:
   statement(s)
```

Note the colon after the boolean-expression. All of the statements must be indented the same amount.




```
    if ( y != 0 ):
       z = ( x / y )
```


If Statement Example

In file `IfExample.py`:

```
def main():
   """ A pretty uninteresting function to illustrate
   if statements. """
   x = int( input("Input an integer or 0 to stop: "))
   if ( x != 0 ):
      print( "You entered", x, ". Thank you!")

main()
```

Would "`if x:`" have worked instead of "`if ( x != 0 ):`"?

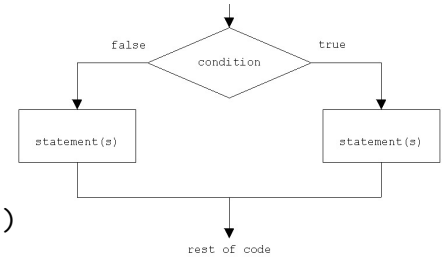
```
> python IfExample.py
Input an integer or 0 to stop: 3
You entered 3. Thank you!
> python IfExample.py
Input an integer or 0 to stop: 0
>
```


Two-way If-else Statements

A two-way **If-else** statement executes one of two actions, depending on the value of a Boolean expression.

General form:

```
if boolean-expression:
   true-case-statement(s)
else:
   false-case-statement(s)
```



Note the colons after the boolean-expression and after the `else`. All of the statements in *both* if and else branches should be indented the same amount.

If-else Statement: Example

In file ComputeCircleArea.py:

```
import math

def main():
   """ Compute the area of a circle, given radius. """
   radius = float( input("Input radius: ") )
   if ( radius >= 0 ):
      area =  math.pi * radius ** 2
      print( "A circle with radius", radius, \
             "has area", round(area, 2) )
   else:
      print( "Negative radius entered.")

main()
```

```
> python ComputeCircleArea.py
Input radius: 4.3
A circle with radius 4.3 has area 58.09
> python ComputeCircleArea.py
Input radius: -3.2
Negative radius entered.
```


Nested If Statements: Leap Year Example

The statements under an `if` can themselves be `if` statements.

For example: Suppose you want to determine whether a particular year is a leap year. The algorithm is as follows:

1. If year is a multiple of 4, then it's a leap year;
2. unless it's a multiple of 100, and then it's not;
3. unless it's also a multiple of 400, and then it is.

Nested If Statements: Is Leap Year?

In file LeapYear.py:

```
def main():
   """ Is entered year a leap year? """
   year = int( input("Enter a year: ") )
   if ( year % 4 == 0 ):
      # Year is a multiple of 4
      if ( year % 100 == 0 ):
         # Year is a multiple of 4
         # and of 100.
         if ( year % 400 == 0 ):
            IsLeapYear = True       # What's true here?
         else:
            IsLeapYear = False      # What's true here?
      else:
         IsLeapYear = True
   else:
      IsLeapYear = False            # What's true here?
   if IsLeapYear:
      print( "Year", year, "is a leap year." )
   else:
      print( "Year", year, "is not a leap year.")
```


Leap Year

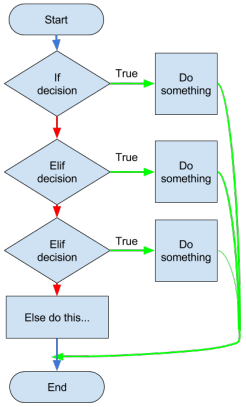
```
> python LeapYear.py
Enter a year: 2000
Year 2000 is a leap year.
> python LeapYear.py
Enter a year: 1900
Year 1900 is not a leap year.
> python LeapYear.py
Enter a year: 2004
Year 2004 is a leap year.
> python LeapYear.py
Enter a year: 2005
Year 2005 is not a leap year.
```

Multiway if-elif-else Statements

If you have multiple options, you can use if-elif-else statements.

General Form:

```
if boolean-expression1:
  statement(s)
elif boolean-expression2:
  statement(s)
elif boolean-expression3:
  ...
else:              # optional
  statement(s)
```



You can have any number of `elif` branches with their conditions. The else branch is optional.

If-elif-else Example

In file `LeapYear3.py`:

```
def main():
   # Is this a leap year
   year = int( input("Enter a year: ") )
   if ( year % 400 == 0 ):
      IsLeapYear = True
   elif ( year % 100 == 0 ):   # what's true here?
      IsLeapYear = False
   elif ( year % 4 == 0 ):     # what's true here?
      IsLeapYear = True
   else:                       # what's true here?
      IsLeapYear = False
   # Print result.
   if IsLeapYear:
      print( "Year", year, "is a leap year." )
   else:
      print( "Year", year, "is not a leap year.")
```

Notice that we could always replace `elif` with nested if-else statements. But this is much more readable. *Be careful with your indentation!*

If-elif-else Example

```
> python LeapYear3.py
Enter a year: 2000
Year 2000 is a leap year.
> python LeapYear3.py
Enter a year: 2004
Year 2004 is a leap year.
> python LeapYear3.py
Enter a year: 1900
Year 1900 is not a leap year.
> python LeapYear3.py
Enter a year: 2005
Year 2005 is not a leap year.
```


Logical Operators

Python has **logical operators** (and, or, not) that can be used to make compound Boolean expressions.

- not : logical negation
- and : logical conjunction
- or : logical disjunction

Operators **and** and **or** are always evaluated using *short circuit evaluation*.

```
( x % 100 == 0 ) and not ( x % 400 == 0 )
```

Short Circuit Evaluation

Notice that (A and B) is False, if A is False; it doesn't matter what B is. *So there's no need to evaluate B, if A is False!*

Also, (A or B) is True, if A is True; it doesn't matter what B is. *So there's no need to evaluate B, if A is True!*

```
>>> x = 13
>>> y = 0
>>> legal = ( y == 0 or x/y > 0 )
>>> print( legal )
True
```

Python doesn't evaluate B if evaluating A is sufficient to determine the value of the expression. *That's important sometimes.*

Leap Years Revisited

Here's an easier way to do our Leap Year computation:

In file `LeapYear2.py`:

```
def main():
    """ Input a year and test whether it's a leap year. """
    year = int( input("Enter a year: ") )

    # What's the logic of this assignment?
    IsLeapYear = ( year % 4 == 0 ) and \
        ( not ( year % 100 == 0 ) or ( year % 400 == 0 ) );

    # Print the answer
    if IsLeapYear:
        print( "Year", year, "is a leap year." )
    else:
        print( "Year", year, "is not a leap year.")

main()
```


Leap Years Revisited

```
> python LeapYear2.py
Enter a year: 2000
Year 2000 is a leap year.
> python LeapYear2.py
Enter a year: 1900
Year 1900 is not a leap year.
> python LeapYear2.py
Enter a year: 2004
Year 2004 is a leap year.
> python LeapYear2.py
Enter a year: 2005
Year 2005 is not a leap year.
```