3. Conditional Execution

Topics:

- Boolean values
- Relational operators
- `if` statements
- The Boolean type

Problem

Assign positive float values to variables **x** and **y** and print " x is greater than y" if x > **y**

Solution:

```
x = float(input('Enter x:'))
y = float(input('Enter y:'))
if x > y:
    print ('x is greater than y')
```

Solution Using If-Else

Repeat the problem and print “y is greater than x” if y > **x**

Solution:

```
x = float(input(‘Enter x:’))
y = float(input(‘Enter y:’))
if x > y:
   print (‘x is greater than y’)
else:
   Print (‘y is greater than x’)
```

The `if-else` Construction

```
if  Boolean expression  :

    Statements to execute if the
        expression if True

else:

    Statements to execute if the
        expression if False
```

This is an example of conditional execution.
The if-else construction is sometimes called "alternative execution"

Even and Odd Problem

Assign positive integer value to variable **x** **then** print "x is even" if it is so, else print "x is odd"

Solution:

```
x = int(input('Enter x:'))
if x%2==0:
    print ('x is even')
else:
    print ('x is odd')
```

String Example

Enter a 5-character string then check if the last character is '`y`', change the '`y`' to '`i`' and add '`es`' Otherwise, just add '`s`'. Assign the result to a variable `t`.

Want: 'carry' to become 'carries'
Use string slicing and concatenation: 'carr' + 'ies'

Solution

```
s = input('Enter a 5 character string:'))
if s[4]=='y':
    t = s[0:4] + 'ies'
else:
    t = s + 's'
print s,t
```

Remember: s[0:4] names the substring comprised of the first 4 characters.

Relational Operators

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Relational Operators in Action

```
x --->   3        y ---> 6
```

```
   x < y       True
 2*x > y       False
   x <= y      True
   x >= y      False
   x == y/2    True
   x != y/2.   False
```

If the expression on the left is a different numerical type then the expression on the right, everything is converted to float.

Boolean Operations with Strings

Comparing for equality…

```
>>> s = 'abc'
>>> s =='abc'
True
>>> s == 'abc '
False
```

Two strings are equal if they have the same length and agree in each position.

Boolean Operations with Strings

Comparing for alphabetical order…

```
>>> s =  'Dog'
>>> s >'Horse'
False
>>> s <  'Horse'
True
    s <  'dog'
```

Alphabetical order. If `s < t` is true then `s` comes before `t` in the "extended dictionary" based on this ordering of characters:

' 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'

Relational Operators in Action

```
x --->  'key'        y --->  'hockey'
```

```
        x < y        False
        x > y        True
'hoc'+x <= y         True
        x >= y       True
        x == y[3:]   True
        x != x+' '   True
```

Comparisons based on alphabetical order.
`x<y` is false because 'key' does not come before 'hockey' in the dictionary.

What if You Have More than Two Alternatives?

For example, given a numerical test score between 0 and 100, print out the letter grade equivalent according to these rules:

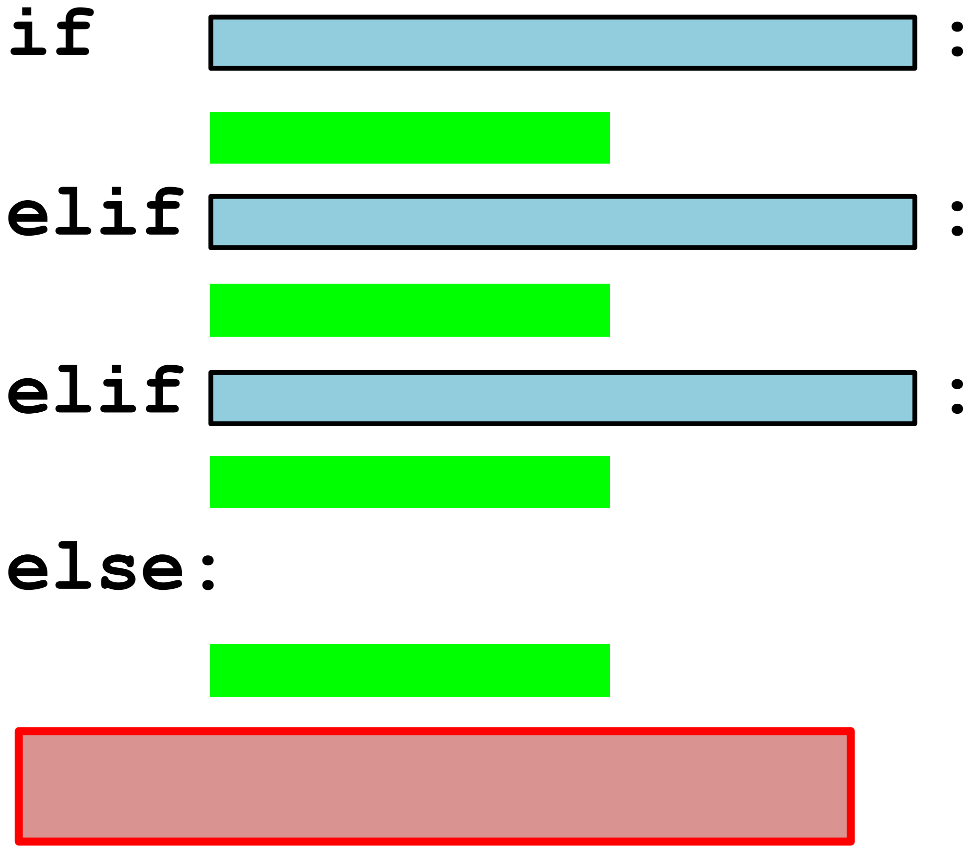
A	90-100
B	80-89
C	70-79
U	<70

The If-Elif-Else Construction

```
x = float(input('Score:'))
if x>=90:
    grade = 'A'
elif x>=80:
    grade = 'B'
elif x>=70:
    grade = 'C'
else:
    grade = 'U'
print (grade)
```

Read "elif" as "else if"

Multiple if-elif With Else



The first green box guarded by a true boolean expression is executed.
If they are all false, then the else's green box is executed.

Boolean Operations

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

It is possible to combine two boolean values (A & B) get a new boolean value.

Boolean Operations

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

It is possible to combine two boolean values (A & B) get a new boolean value.

The and Operation

```
x --->   3     y --->  6     z --->   9
```

```
(x < y) and (x < z)      True
(x > y) and (x < z)      False
(x < y) and (x > z)      False
(x > y) and (x > z)      False
```

The and Operation

🟩	🟨	🟩 and 🟨
True	True	True
True	False	False
False	True	False
False	False	False

Here 🟩 and 🟨 are Boolean-valued expressions

Example

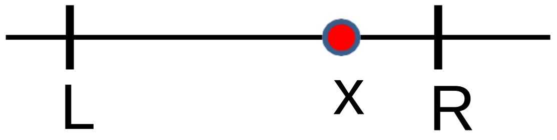
Fact: A length-4 string is a palindrome if the first and last characters are the same and the middle two characters are the same.

```
s = input(‘length-4 string:  ’)

if (s[0]==s[3]) and (s[1]==s[2]):
    print (‘palindrome’)
else:
    print (‘not a palindrome’)
```

Example 2

Fact: x is inside the interval [L,R] if it is no smaller than L and no bigger than R.

```
x  =  int(input('x:  '))
L  =  int(input('L:  '))
R  =  Int(input('R:  '))

if (L<=x) and (x<=R):
   print ('Inside')
else:
   print ('Outside')
```



Equivalent Solutions

```
x   =   int(input('x:   '))
L   =   int(input('L:   '))
R   =   Int(input('R:   '))

if L<=x<=R :
    print ('Inside')
else:
    print ('Outside')
```

The or Operation

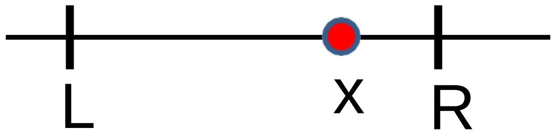
🟩	🟨	🟩 or 🟨
True	True	True
True	False	True
False	True	True
False	False	False

Here 🟩 and 🟨 are Boolean-valued expressions

Example 1

Fact: x is inside the interval [L,R] if it is
no smaller than L and no bigger than R.

```
x  =  int(input('x:  '))
L  =  int(input('L:  '))
R  =  Int(input('R:  '))

if (x<L) or (R<x):
   print ('Outside')
else:
   print ('Inside')
```

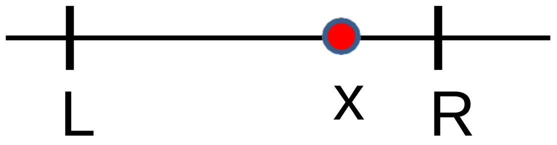


Equivalent Solutions

Fact: x is inside the interval [L,R] if it is no smaller than L and no bigger than R.

```
if (x<L) or (R<x):
    print ('Outside')
else:
    print ('Inside')
```

Often you can arrange a conditional execution in several ways.



The not Operator

🟪	not 🟪
`True`	`False`
`False`	`True`

Here 🟪 is a boolean-valued expression

The not Operation

```
x --->    3          y ---> 6
```

```
not (x < y)      False
not (x > y)      True
```

Summary

1. A Boolean expression evaluates to either `True` or `False`

2. A Boolean expression is made up of comparisons that are either `True` or `False`

3. The `and`, `or`, `not` operations combine Boolean values.

4. Various `if` constructions can be used to organize conditional execution.