

# Relational Operators

---

- These operators compare the value of two ‘expressions’ and returns a Boolean value.

```
>>> A = 10  
>>> B = 10  
>>> A == B  
True
```

```
>>> A = 10.0  
>>> B = 10  
>>> A == B  
True
```

```
>>> A = '10'  
>>> B = 10  
>>> A == B  
False
```

- Beware of comparing across data types, especially when reading values in from command line or files.

# Relational Operators

---

<code>==</code>	equal	True if expressions are equal
<code>!=</code>	not equal	True if expressions are not equal
<code>&gt;</code>	Greater than	True if left is greater than the right
<code>&lt;</code>	Less than	True if left is less than the right
<code>&gt;=</code>	greater than OR equal	
<code>&lt;=</code>	less than OR equal	
<code>is</code>	identity	True if the left is the same object as right
<code>in</code>	contains	True if the object on left is contained in object on right (Useful for finding values in list)

# Assignment Operators

---

- $A += B$  increase A by value of B
  - $A -= B$  decrease A by value of B
  - $A *= B$  multiply A by B and assign value to A
  - $A /= B$  divide A by B and assign value to A
  - $A **= B$  raise value of A to the power of B
  - $A %= B$  modulus of A by B, assigned to A
  - $A //= B$  floor of A divided by B, assigned to A
- 
- String context:
    - $S1 += S2$  add string on right to the one on left
    - $S1 *= A$  Make A copies of S1 and concatenate them to S1

# Boolean Operators

---

Combines two or more statements that return a Boolean value.

A and B      True if both A and B are true

A or B      True if either A or B is true

not A      reverse the Boolean given by A

xor(A,B)      True if only one of A or B is True

A	B	A and B	A or B	Not A	xor(A,B)
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE	FALSE

# General Python Syntax rules

---

- End of line is end of statement
- Statements at the same indentation level are in the same block (e.g., within a loop or condition)
- End of indentation
- Exceptions:
  - Semi colon ; separates statements on the same line

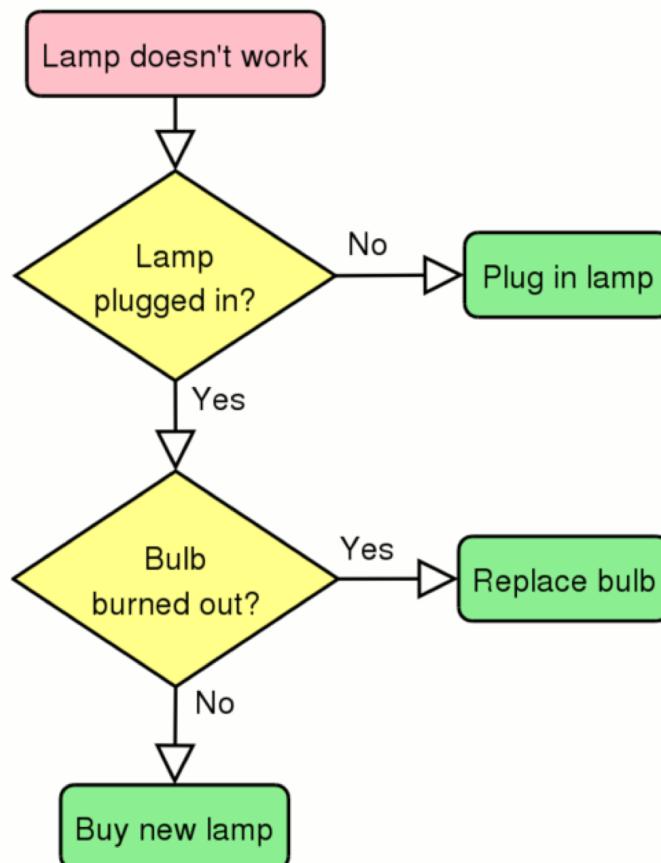
```
>>> A = 10 ; B =20 ; A == B  
False
```

- Single line blocks are allowed without indentation

```
>>> for x in range(5): print x  
...  
0  
1  
2  
3  
4
```

# Branching logic

- Used to implement alternate paths for the logic flow.



<https://upload.wikimedia.org/wikipedia/commons/4/44/LampFlowchart.png>

# If/elif/else statements

---

**if** test1:

    statement 1

**elif** test2:

    statement 2

**else**:

    statement 3

- Both the elif and else blocks are optional.

# If/elif/else statements

---

```
>>> A = 0; B = 10; S1 = 'Hello'  
>>> if (A < B ):  
...     print 'true'  
...  
true  
>>> if (A > B ):  
...     print 'true'  
... else:  
...     print 'false'  
...  
false
```

```
>>> if A == 0 :  
...     print A  
... elif B ==10 :  
...     print B  
... else:  
...     print "Neither match"  
...  
0
```

# Lamp flowchart with if/else

---

```
#!/usr/bin/python

lamp = raw_input("Is the lamp on (yes/no): ")
plugged = raw_input("Is the lamp plugged in (yes/no) : ")
burnt = raw_input("Is the bulb burnt out (yes/no) : ")
if lamp != 'yes':
    if plugged == 'yes':
        if burnt == 'yes':
            print 'replace bulb'
        else:
            print 'replace lamp'
    else:
        print 'plug in lamp'
else: print 'Enjoy the light'
```

Accepts input  
from user, as  
a string

# Truth and Boolean tests in Python

---

- All objects in python have an inherent true or false value.
- Any nonempty object is true.
  - For Integers : Any non-zero number is true
- Zero, empty objects and special object ‘None’ are false.
- Comparisons return the values True or False

# Loops/Iterations

---

- A loop is syntax structure that repeats all the statements within the loop until the exit condition is met.
- Statements in a loop are defined by indenting them relative to the loop start.
- Loop ends when indentation ends.
- Python has two forms of loops: for loop and while loop.
- E.g. >>> for x in range(10)
- E.g. >>>while (A==10)

# while loops

---

- while condition:  
    statement 1  
    statement 2  
    .  
    .
- Most generic form of loop, that checks whether the condition is true at the start of each iteration.
- Expects the condition to become false at some point during the iterations of the loop.
- If condition is never changed, this creates an ‘infinite’ loop. i.e., the program will get stuck in this loop for ever.

# Example while loops

---

```
>>> while (A < B):
...     print A
...     A+=1
...
0
1
2
3
4
5
6
7
8
9
```

```
>>> while (B > A):
...     print B
...     B-=1
...
10
9
8
7
6
5
4
3
2
1
```

```
>>> while S1:
...     print S1
...     S1 = S1[1:]
...
Hello
ello
llo
lo
o
```

# Altering while loops

---

- Normal loop control will execute all statements in block on every iteration. Loop ends only when exit condition is met.
- break statement forces the current loop to exit.
- continue statement skips the rest of the block and goes to the next iteration of the loop.
- pass statement is a placeholder for empty blocks.

# Altering while loops

---

```
>>> A=0
>>> while True:
...     print A
...     if (A >= 5): break
...     A+=1
...
0
1
2
3
4
5
```

```
>>> A=0
>>> while A <= 5 :
...     A+=1
...     if (A == 3):continue
...     print A
...
1
2
4
5
6
```

# for loops

---

- `for item in sequence:`
  - statement 1
  - statement 2
  - .
  - .
- Generic iterator for items in a ordered sequence such as lists, tuples etc.
- On each iteration retrieves one item from the list and assigns it to the variable specified.
- Automatically moves to the next item in the order.
- Value of variable may be altered within the for loop, but change is not made in the list.

# for loops

---

```
>>> L=['Egg','Bacon','Ham']
>>> for item in L:
...     item+='s'
...     print item
...
Eggs
Bacons
Hams
>>> L
['Egg', 'Bacon', 'Ham']
```

```
>>> for x in range(0,100,10):
...     print x
...
0
10
20
30
40
50
60
70
80
90
```

# Looping over Strings and Lists

---

- List is a general sequence object while String is a character sequence object.
- Both can be iterated over by a for loop:

```
>>> L  
['Egg', 'Bacon', 'Ham']  
>>> for x in L:  
...     print x  
...  
Egg  
Bacon  
Ham
```

```
>>> S1="Hello"  
>>> for x in S1:  
...     print x  
...  
H  
e  
l  
l  
o
```

# Looping over lists with and without index

---

- Looping with an index allows accessing the item within the list and changing it.

```
>>> for x in L:  
...     x+='s'  
...     print x  
...  
Eggs  
Bacons  
Hams  
>>> L  
['Egg', 'Bacon', 'Ham']
```

```
>>> for x in range(len(L)):  
...     L[x]+='s'  
...     print L[x]  
...  
Eggs  
Bacons  
Hams  
>>> L  
['Eggs', 'Bacons', 'Hams']
```

# Looping over Tuples and Dictionaries

---

```
>>> T = (1,'a',45,23.45,'String')
>>> for x in T: print x
...
1
a
45
23.45
String
```

```
>>> myDict = {'a':'1','b':'2','c':'3'}
>>> for key in myDict:
...     print(key,' ==> ', myDict[key])
...
('a', ' ==> ', '1')
('c', ' ==> ', '3')
('b', ' ==> ', '2')
```

```
>>> for (key,value) in myDict.items():
...     print(key,' ==> ', value)
...
('a', ' ==> ', '1')
('c', ' ==> ', '3')
('b', ' ==> ', '2')
```

# Nested Loops

---

- Loops can be nested just like the if/else statements.
- Indentation is again the key to creating nested loops.
- In a 2 level nested loop with x iterations on the outer loop and y iterations in the inner loop:
  - All statements in the outer loop will be executed x times
  - All statements in the inner loop will be executed  $x*y$  times