The `for` loop

- **`for` loop**: Repeats a set of statements over a group of values.
  - Syntax:

```
for variableName in groupOfValues:
    statements
```

    - We indent the statements to be repeated with tabs or spaces.
    - ***variableName*** gives a name to each value, so you can refer to it in the ***statements***.
    - ***groupOfValues*** can be a range of integers, specified with the `range` function.

  - Example:

```
for x in range(1, 6):
    print x, "squared is", x * x
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

range

- The `range` function specifies a range of integers:
  - `range(`***start***`, `***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)
  - It can also accept a third value specifying the change between values.
    - `range(`***start***`, `***stop***`, `***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***
  - Example:

```
for x in range(5, 0, -1):
    print x
print "Blastoff!"
```

Output:

```
5
4
3
2
1
Blastoff!
```

  - **Exercise:** How would we print the "99 Bottles of Beer" song?

Cumulative loops

- Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a *cumulative sum*.

```
sum = 0
for i in range(1, 11):
    sum = sum + (i * i)
print "sum of first 10 squares is", sum

Output:
sum of first 10 squares is 385
```

- **Exercise:** Write a Python program that computes the factorial of an integer.

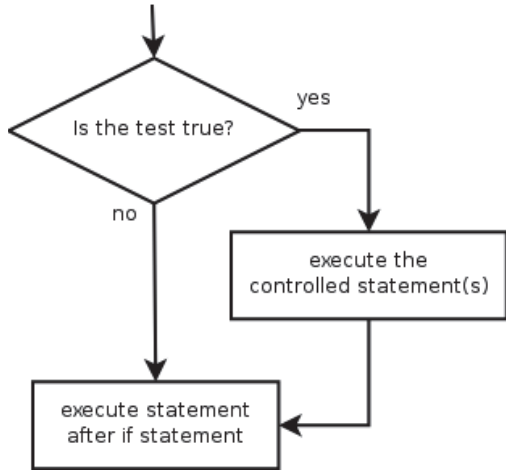
if

- **`if` statement**: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
  - Syntax:

```
if condition:
    statements
```

- Example:

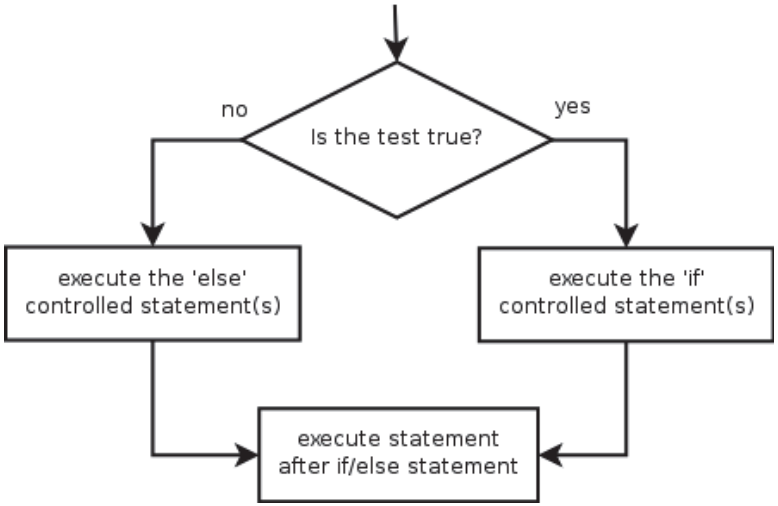
```
gpa = 3.4
if gpa > 2.0:
    print "Your application is accepted."
```



if/else

- **`if/else` statement**: Executes one block of statements if a certain condition is True, and a second block of statements if it is False.
  - Syntax:

```
if condition:
    statements
else:
    statements
```

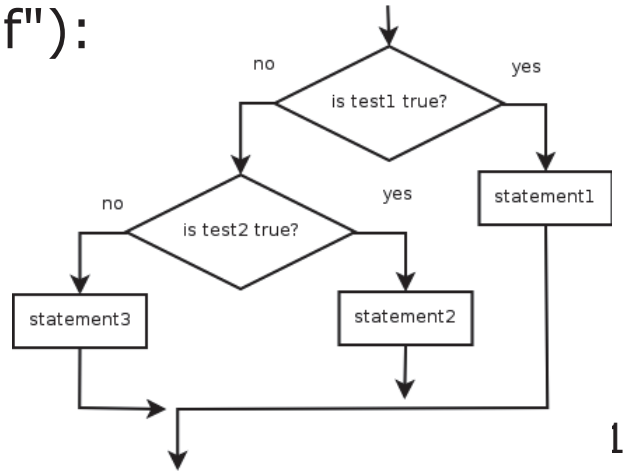


- Example:

```
gpa = 1.4
if gpa > 2.0:
    print "Welcome to Mars University!"
else:
    print "Your application is denied."
```

- Multiple conditions can be chained with `elif` ("else if"):

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```



Example of If Statements

```
import math
x = 30
if x <= 15 :
    y = x + 15
elif x <= 30  :
    y = x + 30
else :
    y = x
print 'y = ',
print math.sin(y)
```

In file ifstatement.py

```
>>> import ifstatement
y =  0.999911860107
>>>
```

In interpreter

while

- **`while` loop**: Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)
- Syntax:

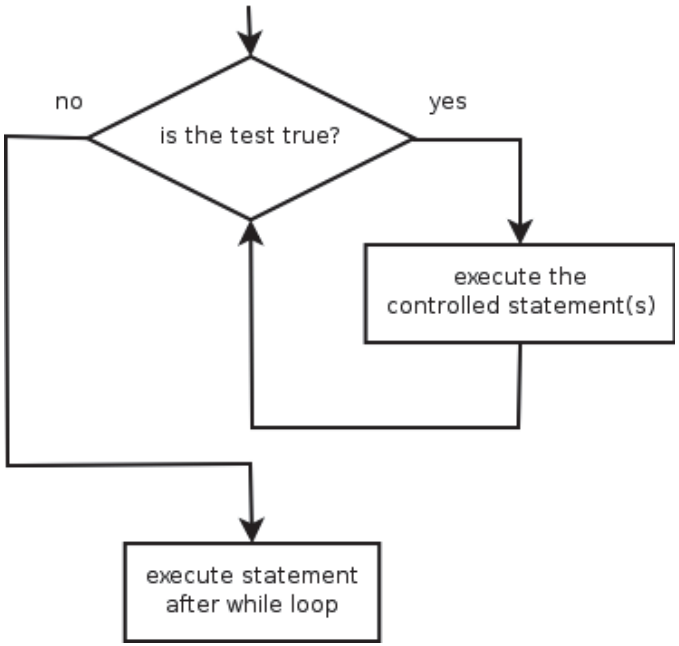
```
while condition:
    statements
```

- Example:

```
number = 1
while number < 200:
    print number,
    number = number * 2
```

  - Output:

```
1 2 4 8 16 32 64 128
```



While Loops

```
x = 1
while x < 10 :
    print x
    x = x + 1
```

- In whileloop.py

```
>>> import whileloop
1
2
3
4
5
6
7
8
9
>>>
```

- In interpreter

Logic

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
==	equals	`1 + 1 == 2`	`True`
!=	does not equal	`3.2 != 2.5`	`True`
<	less than	`10 < 5`	`False`
>	greater than	`10 > 5`	`True`
<=	less than or equal to	`126 <= 100`	`False`
>=	greater than or equal to	`5.0 >= 5.0`	`True`

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`

- **Exercise:** Write code to display and count the factors of a number.

Loop Control Statements

break	Jumps out of the closest enclosing loop
continue	Jumps to the top of the closest enclosing loop
pass	Does nothing, empty statement placeholder

More Examples For Loops

- Similar to perl for loops, iterating through a list of values

forloop1.py

```
for x in [1,7,13,2]:
    print x
```

```
%python forloop1.py
1
7
13
2
```

forloop2.py

```
for x in range(5) :
    print x
```

```
% python forloop2.py
0
1
2
3
4
```

range(N) generates a list of numbers [0,1, …, n-1]