1 For loops

We can use a **for** statement to iterate (or loop) over the characters of a string. For each character, we execute a set of statements called the loop body. Example:

```
name = "bud"
for ch in name:
    print "-->",
    print ch
print "--> !"
```

The above code prints this:

```
--> b
--> u
--> d
--> !
```

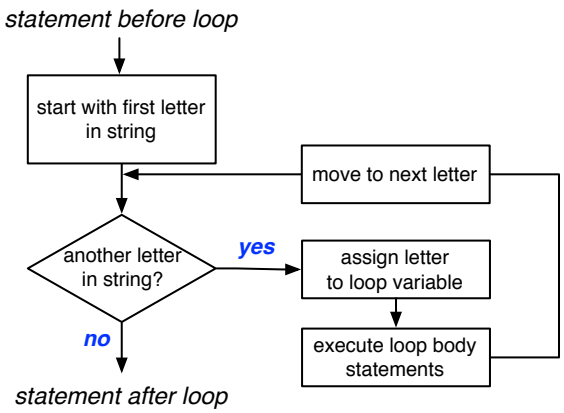
The indented statements are the **loop body**. Since the string `name` has three characters, the body is executed three times, once per character. The variable `ch` is the **loop variable**. Each time through the loop, it is assigned to the next character in the string `name`. Python relies on indentation to figure out where the loop body ends. For instance, since `print "--> !"` is not indented, it is only executed once after the loop has finished.

The loop variable can be named whatever you want. This program does the same thing:

```
name = "bud"
for banana in name:
    print "-->",
    print banana
print "--> !"
```

However, `banana` is a lousy variable name. Use appropriate names in your programs!

This diagram illustrates how python executes a for loop:



2 Accumulator pattern

The **accumulator pattern** is a general strategy for completing a task using a for loop. We can use it to complete tasks involving strings where we accumulate a result by processing each character of the string one at a time. Here are some key steps in applying the accumulator pattern:

1. Answer these key questions:
   (a) What do you want to compute? What is the final result?
   (b) Assuming the final result is a data value, what is its type (`int`, `float`, `str`, etc.)?
   (c) How can we build up the final result by processing one letter at a time?
2. Initialize an accumulator variable: make sure it has the right type!
3. Each time through the loop, update the accumulator variable.
4. When the loop finishes, the accumulator variable should contain the final result.

Example: ask the user for his/her name and calculate its length. Answers to the key questions: 1.a) the final result is the number of characters in the name, 1.b) type is `int`, 1.c) we can loop over the letters of the string and keep count.

```
name = raw_input("What is your name? ")
count = 0   # initialize accumulator variable
for letter in name:
    count = count + 1 # update accumulator (add 1 for each letter)
# when loop is over count is equal to name's length
print "Your name has", count, "letters."
```

Example: print a copy of the name with each letter duplicated. If the user types "bud", then your program should print "bbuudd." Answers to the key questions: 1.a) name with duplicate letters, 1.b) type is `str`, 1.c) we make a new string and as we loop over the characters in the string, we concatenate two copies of the character to the end of our string.

```
dup_name = ''   # initialize accumulator variable to be empty string
for letter in name:
    dup_name = dup_name + letter + letter # update accumulator
# when loop is over, dup_name has final result
print "Your name in duplicate", dup_name
```

3 Exercises

1. Write a short program that asks the user for their name and then prints a copy of the name that is *both* duplicated *and* reversed. For example, if the user enters `"bud"`, it would print `"dduubb"`.

Solution:

```
name = raw_input("Please enter your name: ")
name2 = ''   # initialize accumulator variable to be empty string
for letter in name:
    name2 = letter + letter + name2  # update accumulator
# when loop is over, name2 has final result
print "Your name reversed and duplicated", name2
```

2. Same idea but print a *palindrome* version of name (palindromes read the same forwards and backwards). For example, if the user enters `"bud"`, it would print `"buddub"`.

Solution:

```
name = raw_input("Please enter your name: ")
name2 = ''   # initialize accumulator variable to be empty string
for letter in name:
    name2 = letter + name2 # update accumulator
name2 = name + name2
# when loop is over, name2 has final result
print "Your palindrome name is", name2
```