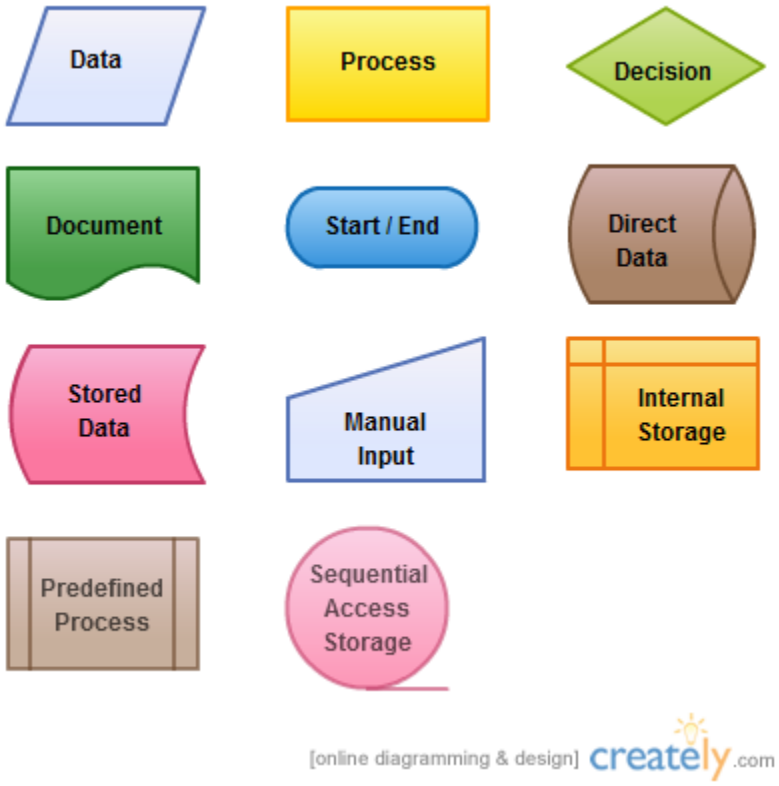
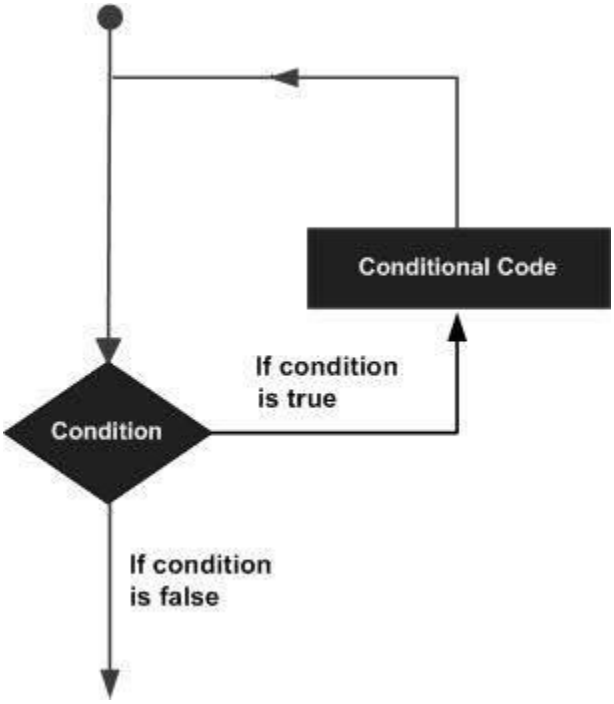
Loops, Logic and Data

Flow Chart Logic elements:



To conclude, a loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages —

Loop Architecture



Unknown number of times:

"Ask the User to Guess a pre-determined number between 1 and 100". You have no way of knowing how many guesses it will take.

"Randomly look in an array for a given value." You have no way of knowing how many tries it will take to find the actual value.

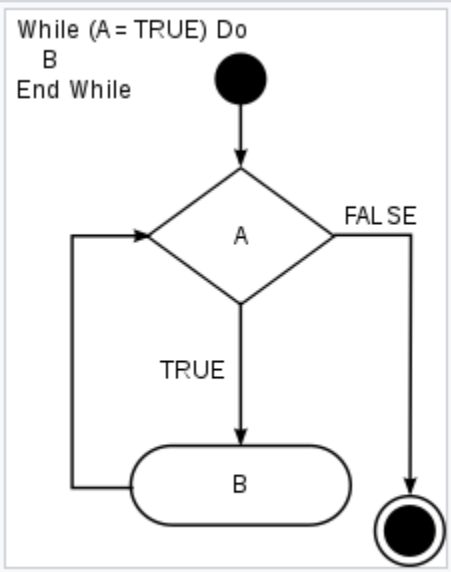
Note: this is a made-up example, because you would never randomly look into an array to find a value. You would always start at the front of the array and look one element at a time until you found the item or got to the end of the array.

Known number of times:

Compute the average grade of the class. While you (the programmer) might not know how many grades exist in the class, the computer will know. Usually this is accomplished by using the "length" function on an array.

Print the odd numbers from 1 to 1001.

Search a list (array) of numbers for the biggest grade. Again, the computer "knows" how many grades there are, so a for loop is appropriate.



While loop flow diagram

Python Code

```
counter = 5                  # Set the value to 5
factorial = 1                # Set the value to 1

while counter > 0:           # While counter(5) is greater than 0
   factorial *= counter      # Set new value of factorial to counter.
   counter -= 1              # Set the counter to counter - 1.

print(factorial)             # Print the value of factorial.
```

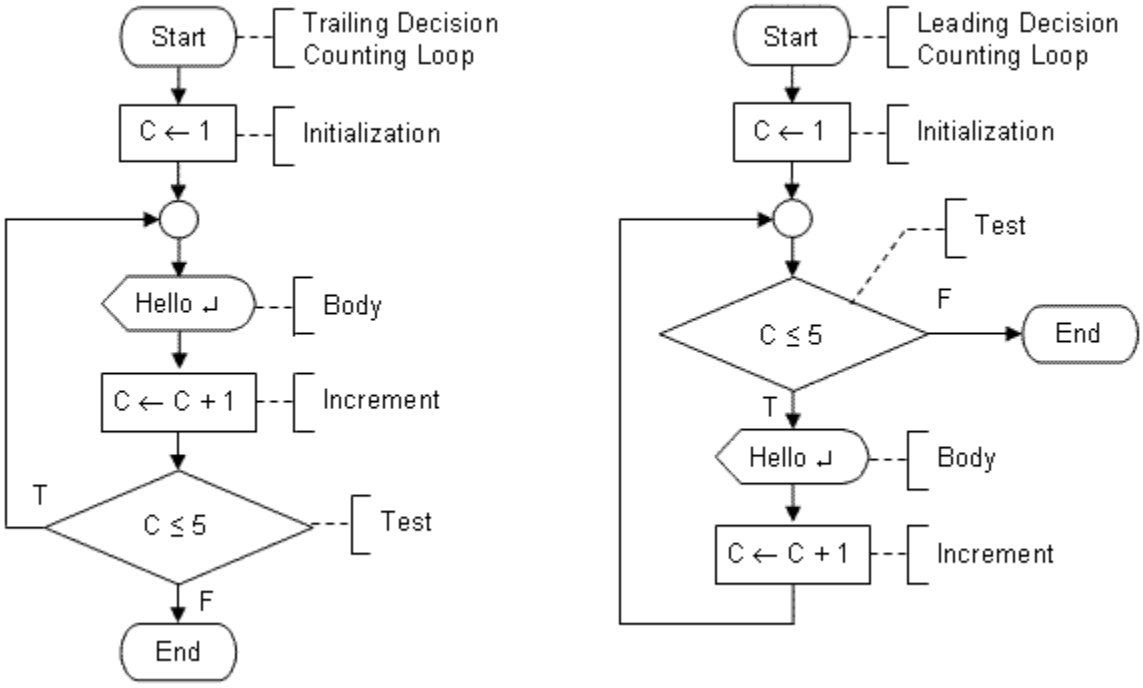
Non-terminating while loop:

```
while True:
   print("Help! I'm stuck in a loop!")
```

Sequential programming is not an efficient way of writing a program,

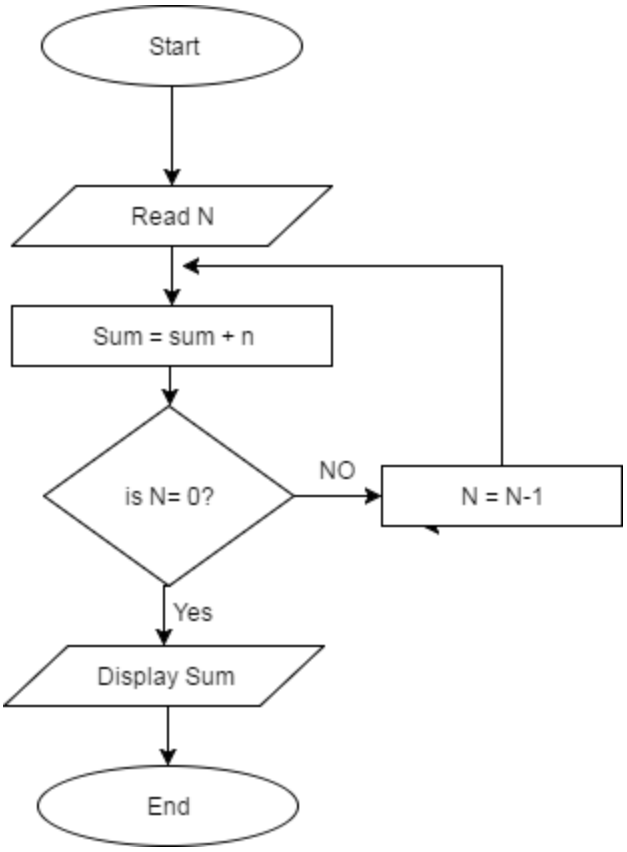
sometimes execution of a program may need to be repeated.

Structure of For Loop Flowchart Case I (Known count)



Flowchart for sum of n numbers Case II (Unknown count)

Here in this an example, we will see how to draw a flowchart to find the sum of any “n” numbers. Since “n” is unknown, it has to be taken as input from the user.



Case I: Yes(the value of “n” has reached 0): In this case, the “sum” already has the sum of numbers from “n” to 1. So, next step is to print the display the result.

Case II: NO(the value of “n” is still > 0): In this case we decrease the value of “n” by 1, and add it’s new value to sum(sum = sum+n) and the steps continue until the value of n becomes 0.

Python While Loops

Python Loops

Python has two primitive loop commands:

- `while` loops
- `for` loops

The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Try it Yourself »

Note: remember to increment i, or else the loop will continue forever.

The `while` loop requires relevant variables to be ready, in this example we need to define an indexing variable, `i`, which we set to 1.

The break Statement

With the `break` statement we can stop the loop even if the while condition is true:

Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Try it Yourself »

The continue Statement

With the `continue` statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Try it Yourself »

The else Statement

With the `else` statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Try it Yourself »

Exercise:

Print `i` as long as `i` is less than 6.

```
i = 1
[ ] i < 6[ ]
  print(i)
  i += 1
```

Submit Answer »

Start the Exercise