Elements of Flow Control

Flow control statements often start with a part called the *condition*, and all are followed by a block of code called the *clause*.

Conditions

- The Boolean expressions you've seen so far could all be considered conditions, which are the same thing as expressions; *condition* is just a more specific name in the context of flow control statements.
- Conditions always evaluate down to a Boolean value, True or False.
- A flow control statement decides what to do based on whether its condition is True or False, and almost every flow control statement uses a condition.

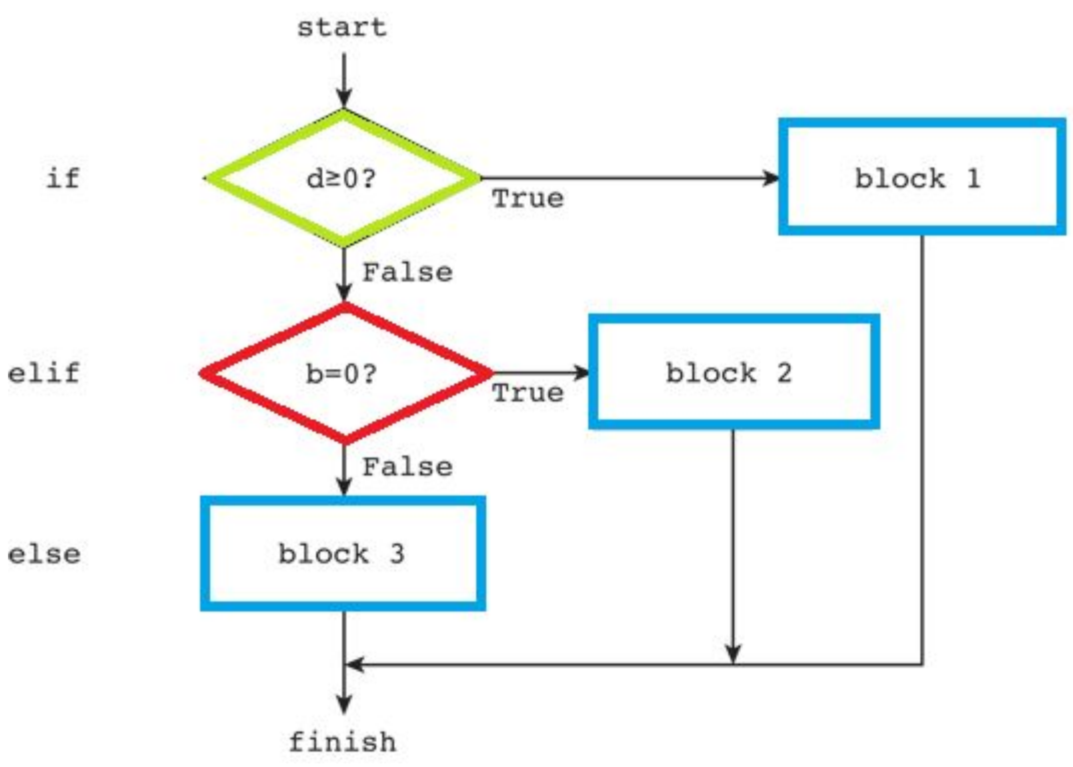
Blocks of Code

Lines of Python code can be grouped together in *blocks*. You can tell when a block begins and ends from the indentation of the lines of code. There are three rules for blocks.

- Blocks begin when the indentation increases.
- Blocks can contain other blocks.
- Blocks end when the indentation decreases to zero or to a containing block's indentation.

- A Python program is constructed from code blocks.
- A *block* is a piece of Python program text that is executed as a unit.
- The following are blocks: a module, a function body, and a class definition.
- Each command typed interactively is a block. A script file (a file given as standard input to the interpreter or specified as a command line argument to the interpreter) is a code block.

Execution



- **If statement**
- **Elif**
- **Else**

Syntex

```
if condition :
      statement(s)


elif condition :
      statement(s)


else :
      statement(s)
```

Example

```
num=int(input())

-10

if num > 0:
    print("Output: A")
elif num < 0:
    print(" Output: B")
else:
    print("Output:O")


Output: B
```

```
>>> x=int(input())
15
>>> if x % 2 == 0:
      print("number is even")
    else :
      print("number is odd")


number is odd
```

```
a=input()

if a == "coding":
    print("coding blocks")
elif a == "python":
    print("Python is the best!")
elif a == "pizza":
    print("can die for")
else:
    print("null")
```

```
Coding     #input
coding blocks    #output
```

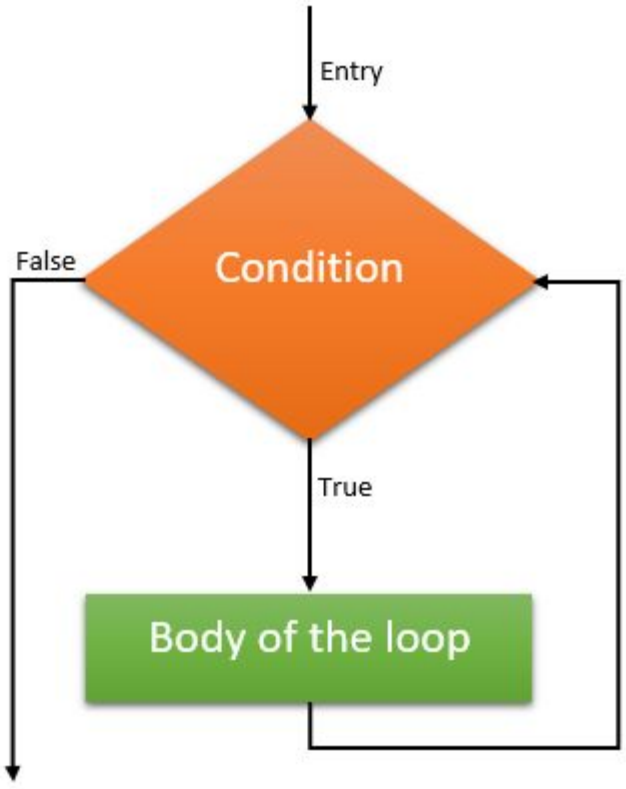
Nested if conditions

```
a="food"
foodName= "pizza"


if a == "food":
    if foodName == "pizza": #nested
        print("Italian")

else:
    print("null")


Italian
```

Looping constructs

Repeat your actions using loops!

The **while** loop



Syntex

```
while condition :
        statement(s)
```

- The while statement is used for repeated execution as long as an expression is true:
- This repeatedly tests the expression and, if it is true, executes the first suite;
- If the expression is false (which may be the first time it is tested) the suite of the else clause, if present, is executed and the loop terminates.

```
a=0
while a < 10:
    print(a)
    a+=1

>>>
 RESTART:
C:/Users/RAJAT/AppData/Local/Programs/Python/Python36-32/python.py
0
1
2
3
4
5
6
7
8
9
```

The break statement

A break statement executed in the first suite terminates the loop without executing the else clause's suite.

```
i = 1
while i < 20:
  print(i)
  if i == 7:
    break
  i += 1

RESTART: C:/Users/RAJAT/AppData/Local/Programs/Python/Python36-32/python.py
1
2
3
4
5
6
7
```

The for loop

The for statement in Python differs a bit from what you may be used to in C or C++. Rather than giving the user the ability to define both the iteration step and halting condition (as C).
Python for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

- The for keyword
- A variable name
- The in keyword : this tests whether or not a sequence contains a certain value.
- A call to the range() method with up to three integers passed to it
- A colon
- Starting on the next line, an indented block of code (called the for clause)
- you can use continue and break statements only inside while and for loops.
- Range fn accepts 1 , 2 or 3 args.

```
a=[1,2,3,4,5,6,7,8]
for i in a:
    if i % 2 == 0:
        print("number in the list is even")
    else:
        print("number in the list is odd")
number in the list is odd #1
number in the list is even #2
number in the list is odd #3
number in the list is even #4
number in the list is odd #5
number in the list is even #6
number in the list is odd #7
number in the list is even #8
```

The continue statement: it continues the next iterations

```
b=[1,2,3,4,5,6,7,8]
for i in b:
     if i % 2 == 0:
         print("number is even",i)
         continue
     print("number exist", i)

number exist 1
number is even 2
number exist 3
number is even 4
number exist 5
number is even 6
number exist 7
number is even 8
```