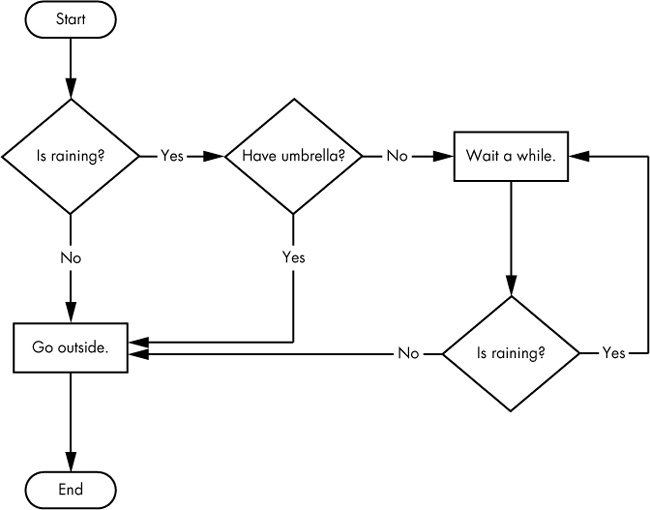
# Flow Control

A program is just a series of instructions.

The real strength of programming isn’t just running (or executing) one instruction after another instruction.

Based on how the expressions evaluate, the program can decide to skip instructions, repeat them, or choose one of several instructions to run. In fact, you almost never want your programs to start from the first line of code and simply execute every line, straight to the end.

Flow control statements can decide which Python instructions to execute under which conditions.



**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 expressions which returns either True or False value are known as conditions.

*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.

1. Blocks begin when the indentation increases.
2. Blocks can contain other blocks.
3. Blocks end when the indentation decreases to zero or to a containing block’s indentation.

Blocks are easier to understand by looking at some indented code.

**if Statements**

The most common type of flow control statement is the if statement.

An if statement’s clause (that is, the block following the if statement) will execute if the statement’s condition is True. The clause is skipped if the condition is False.

In plain English, an if statement could be read as, “If this condition is true, execute the code in the clause.” In Python, an if statement consists of the following:

* The if keyword
* A condition (that is, an expression that evaluates to True or False)
* A colon
* Starting on the next line, an indented block of code (called the if clause)

syntax

if condition: stmt

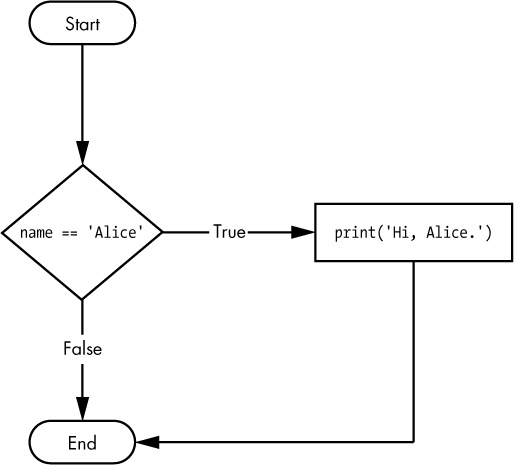
(or)

if condition:

stmt1

stmt2

-----



Eg:

Name=raw\_input(“enter name”)

if Name == 'Alice':

print('Hi, Alice.')

print “Hello World”

## else Statements

An if clause can optionally be followed by an else statement. The else clause is executed only when the if statement’s condition is False. In plain English, an else statement could be read as, “If this condition is true, execute this code. Or else, execute that code.” An else statement doesn’t have a condition, and in code, an else statement always consists of the following:

* The else keyword
* A colon
* Starting on the next line, an indented block of code (called the else clause)

Syntax

If condition:

Statement1

Statement2

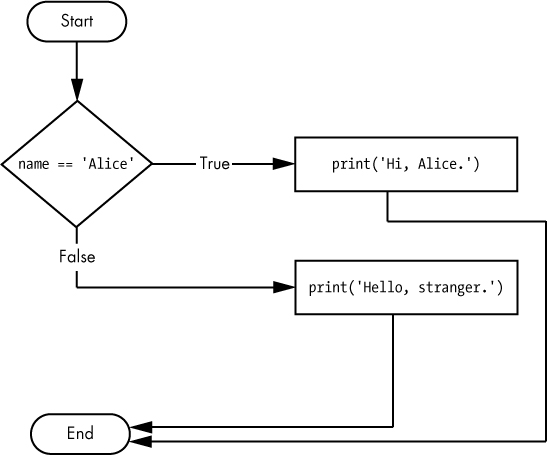
----------------

Else:

Statement1

Statement2

----------------



Eg:

name = raw\_input('enter name')

if name == 'Alice':

print('Hi, Alice.')

else:

print('Hello, stranger.')

## elif Statements

While only one of the if or else clauses will execute, you may have a case where you want one ofmany possible clauses to execute. The elif statement is an “else if” statement that always follows anif or another elif statement. It provides another condition that is checked only if any of the previous conditions were False. In code, an elif statement always consists of the following:

* The elif keyword
* A condition (that is, an expression that evaluates to True or False)
* A colon
* Starting on the next line, an indented block of code (called the elif clause)

Syntax

If condition:

Statement1

Statement2

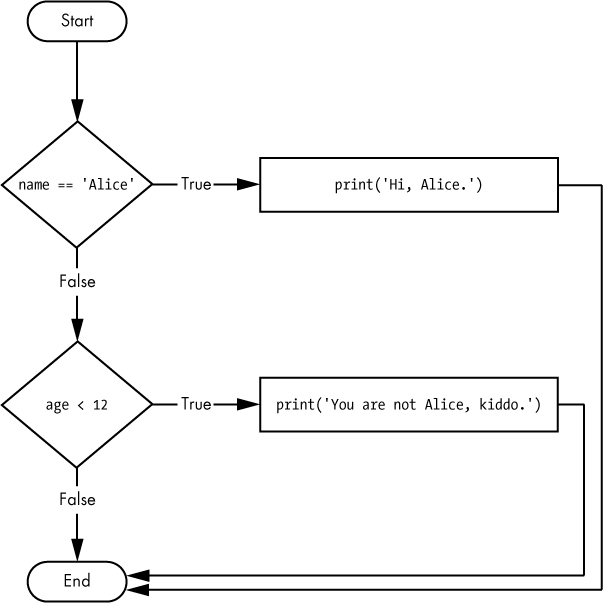
-----------------

Elif condition:

Statement1

Statement2

-----------------



Eg1:

name =raw\_input( 'enter name')

age =input(“enter age”)

if name == 'Alice':

print('Hi, Alice.')

elif age < 12:

print('You are not Alice, kiddo.')

The order of the elif statements does matter.

Remember that the rest of the elif clauses are automatically skipped once a True condition has been found.

Eg2:

name = raw\_input('enter name')

age = input(‘enter age’)

if name == 'Alice':

print('Hi, Alice.')

elif age < 12:

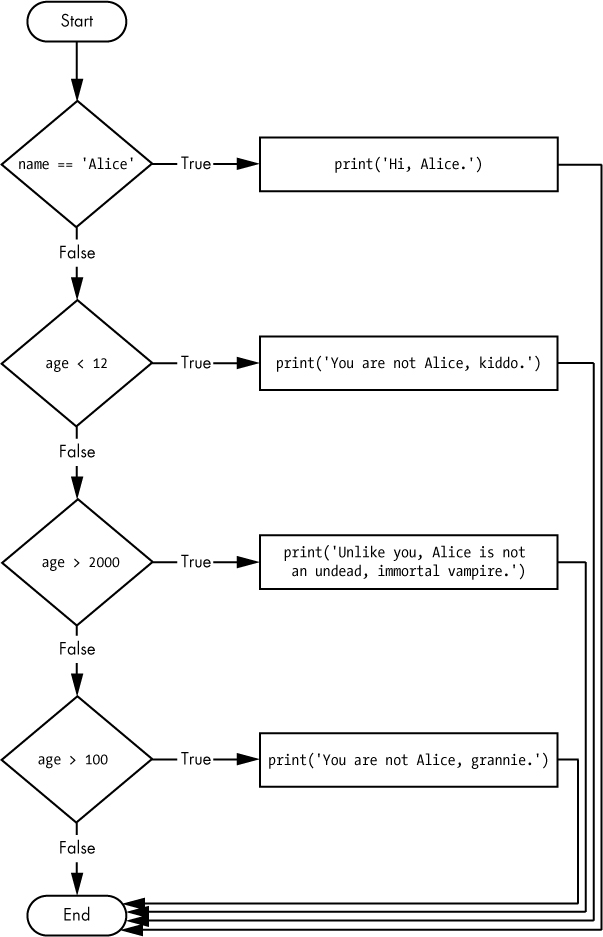
print('You are not Alice, kiddo.')

elif age > 2000:

print('Unlike you, Alice is not an undead, immortal vampire.')

elif age > 100:

print('You are not Alice, grannie.')

Say the age variable contains the value 3000. You might expect the code to print the string 'Unlike you, Alice is not an undead, immortal vampire.'. However, because the age > 100 condition is True (after all, 3000 is greater than 100) , the string'You are not Alice, grannie.' is printed, and the rest of the elif statements are automatically skipped. Remember, at most only one of the clauses will be executed, and for elif statements, the order matters.

Eg3:

name = raw\_input('Enter name’)

age = input(‘enter age’)

if name == 'Alice':

print('Hi, Alice.')

elif age < 12:

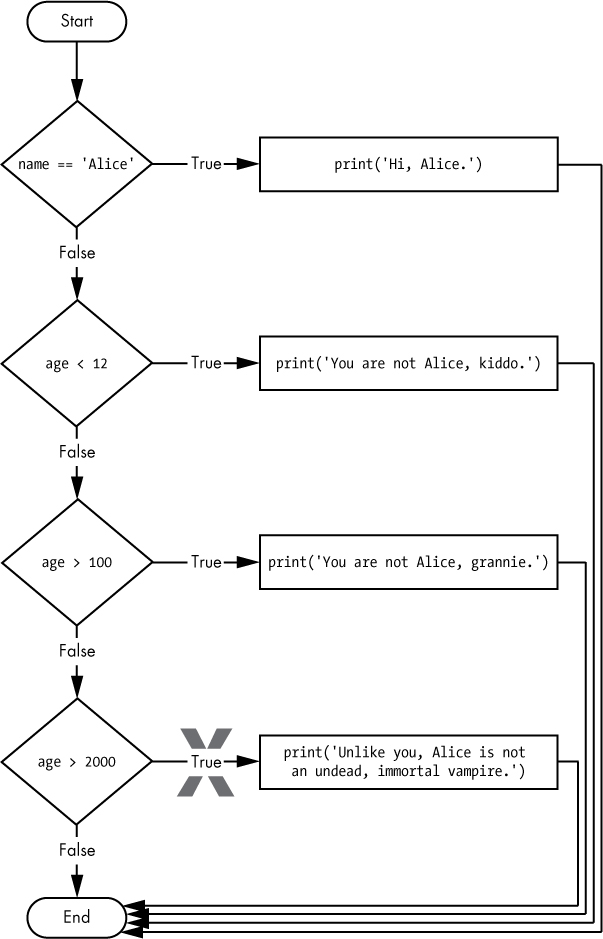
print('You are not Alice, kiddo.')

elif age > 100:

print('You are not Alice, grannie.')

elif age > 2000:

print('Unlike you, Alice is not an undead, immortal vampire.')



Optionally, you can have an else statement after the last elif statement. In that case, it is guaranteed that at least one (and only one) of the clauses will be executed. If the conditions in every if and elifstatement are False, then the else clause is executed.

Eg4:

name =raw\_input( 'Enter name')

age =input(‘enter age’)

if name == 'Alice':

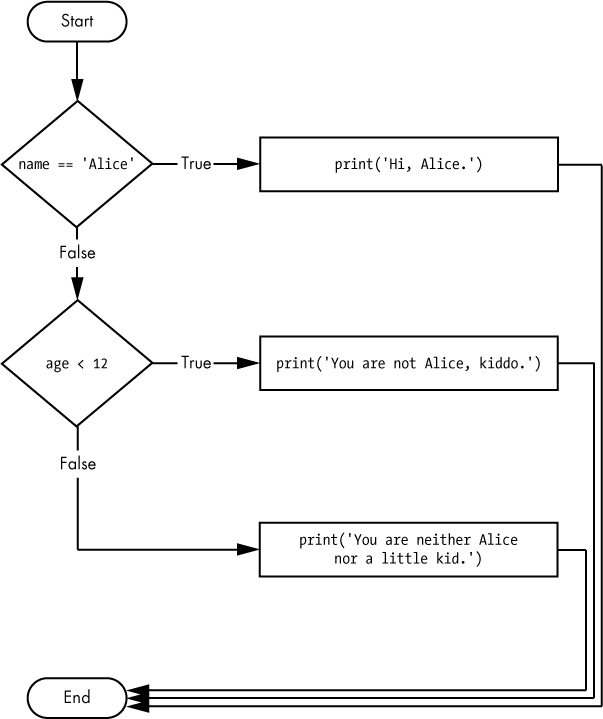
print('Hi, Alice.')

elif age < 12:

print('You are not Alice, kiddo.')

else:

print('You are neither Alice nor a little kid.')



## while Loop Statements

You can make a block of code execute over and over again with a while statement. The code in a while clause will be executed as long as the while statement’s condition is True. In code, a while statement always consists of the following:

* The while keyword
* A condition (that is, an expression that evaluates to True or False)
* A colon
* Starting on the next line, an indented block of code (called the while clause)

You can see that a while statement looks similar to an if statement. The difference is in how they behave. At the end of an if clause, the program execution continues after the if statement. But at the end of a while clause, the program execution jumps back to the start of the while statement. Thewhile clause is often called the while loop or just the loop.

Syntax

While condition:

Statement1

Statement2

---------------

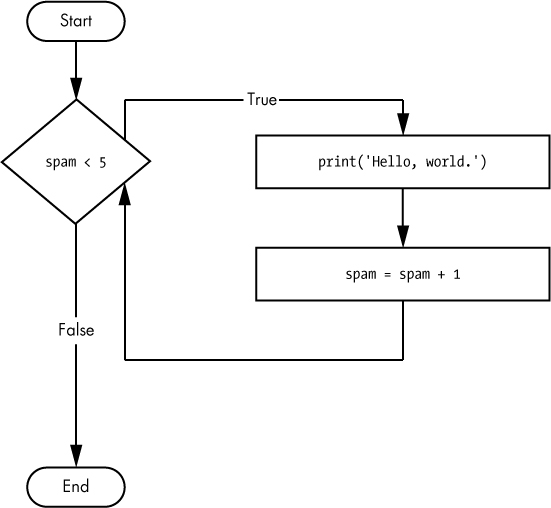
Eg1:

spam = 0

while spam < 5:

print('Hello, world.')

spam = spam + 1



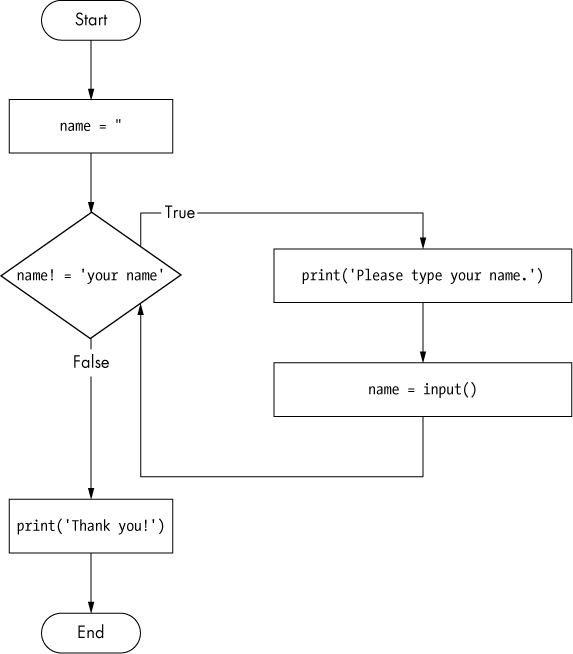
Eg2:

name = ''

while name != 'your name':

name =raw\_ input(‘enter name’)

print('Thank you!')



First, the program sets the name variable to an empty string. so that the name != 'your name' condition will evaluate to True and the program execution will enter the while loop’s clause .

The code inside this clause asks the user to type their name, which is assigned to the name variable . Since this is the last line of the block, the execution moves back to the start of the while loop and re evaluates the condition. If the value in name is not equal to the string 'your name', then the condition is True, and the execution enters the while clause again.

But once the user types **your name**, the condition of the while loop will be 'your name' != 'your name', which evaluates to False. The condition is now False, and instead of the program execution re entering the while loop’s clause, it continues running the rest of the program .

If you never enter **your name**, then the while loop’s condition will never be False, and the program will just keep asking forever. Here, the raw\_input() call lets the user enter the right string to make the program move on. In other programs, the condition might never actually change, and that can be a problem.

## break Statements

There is a shortcut to getting the program execution to break out of a while loop’s clause early. If the execution reaches a break statement, it immediately exits the while loop’s clause. In code, a break statement simply contains the break keyword.

Eg:

while True:

name =raw\_ input('Please type your name.')

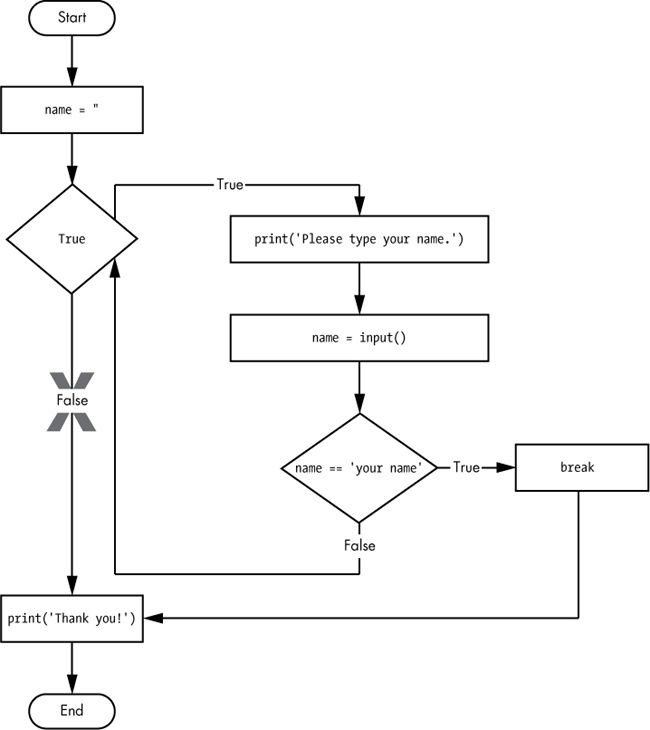
if name == 'your name':

break

print('Thank you!')

The first line creates an infinite loop, it is a while loop whose condition is always True. (The expression True, after all, always evaluates down to the value True.) The program execution will always enter the loop and will exit it only when a break statement is executed. (An infinite loop that never exits is a common programming bug.)

this program asks the user to type **your name** . Now, while the execution is still inside the while loop, an if statement gets executed to check whether name is equal to your name. If this condition is True, the break statement is run and the execution moves out of the loop to print('Thank you!') . Otherwise the if statement’s clause with the break statement is skipped, which puts the execution at the end of the while loop. At this point, the program execution jumps back to the start of the while statement to recheck the condition. Since this condition is merely the True Boolean value, the execution enters the loop to ask the user to type **your name** again.



## continue Statements

Like break statements, continue statements are used inside loops. When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop and re evaluates the loop’s condition. (This is also what happens when the execution reaches the end of the loop.)

while True:

name = raw\_input(‘who are you?’)

if name != 'Joe':

continue

password =raw\_ input('Hello, Joe. What is the password? ')

if password == 'swordfish':

break

print('Access granted.')

If the user enters any name other than  Joe , the continue statement causes the program execution to jump back to the start of the loop. When it re evaluates the condition, the execution will always enter the loop, since the condition is simply the value True. Once they make it past that if statement, the user is asked for a password . If the password entered is swordfish, then the break statement is run, and the execution jumps out of the while loop to print Access granted . Otherwise, the execution continues to the end of the while loop, where it then jumps back to the start of the loop.

