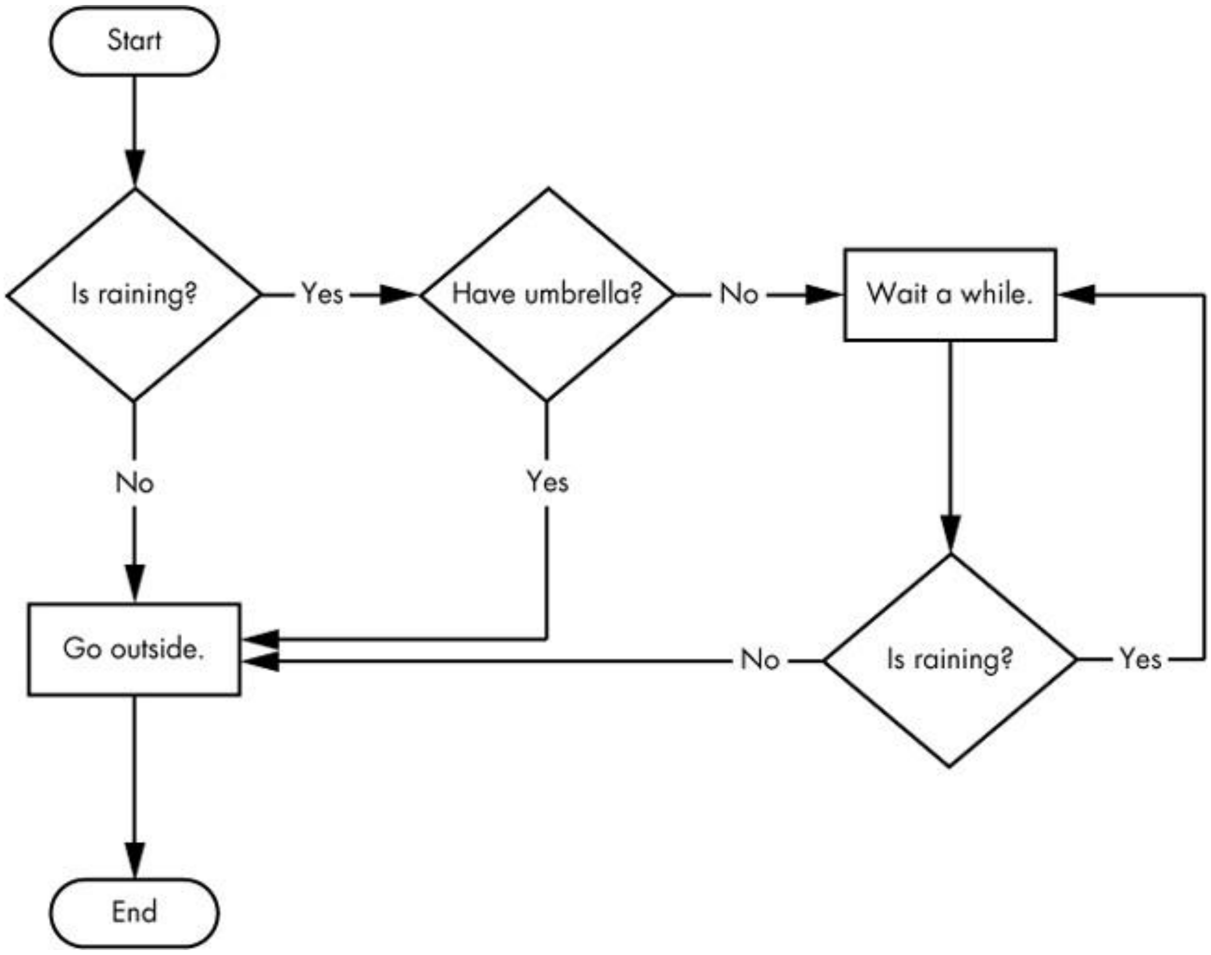
Based on how the expressions evaluate, the program can decide to skip instructions, repeat them, or choose one of several instructions to run. In fact, you almost never want your programs to start from the first line of code and simply execute every line, straight to the end. Flow control statements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

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.

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.

"If this condition is true, execute the code in the clause."

```
name = 'Mary'

password = 'swordfish'

if name == 'Mary':

    print('Hello Mary')

    if password == 'swordfish':

        print('Access granted.')

    else:

        print('Wrong password.')
```

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.

an elsestatement could be read as, "If this condition is true, execute this code. Or else, execute that code."

```
name = 'Bob'

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 of many possible clauses to execute. The elif statement is an “else if” statement that always follows an if or another elif statement. It provides another condition that is checked only if all of the previous conditions were False.

```
name = 'Bob'

age = 5

if name == 'Alice':

    print('Hi, Alice.')

elif age < 12:

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

The elif clause executes if age < 12 is True and name == 'Alice' is False. However, if both of the conditions are False, then both of the clauses are skipped. It is notguaranteed that at least one of the clauses will be executed. When there is a chain of elif statements, only one or none of the clauses will be executed. Once one of the statements’ conditions is found to be True, the rest of the elif clauses are automatically skipped.

```
name = 'Dracula'

age = 4000

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.')
```

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.

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.

If code:

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
o/p: Hello, world.
```

While code:

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
o/p:
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
```

The code with the if statement checks the condition, and it prints Hello, world.only once if that condition is true. The code with the while loop, on the other hand, will print it five times. It stops after five prints because the integer in spam is incremented by one at the end of each loop iteration, which means that the loop will execute five times before spam < 5 is False.

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.

```
while True:

    print('Please type your name.')

    name = input()

    if name == 'Python':

        break

print('Thank you!')
```

The first line creates an *infinite loop*; it is a while loop whose condition is always True. The program execution will always enter the loop and will exit it only when a break statement is executed.

This program asks the user to type a name. Now, however, while the execution is still inside the while loop, an if statement gets executed to check whether name is equal to Python. 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 Python 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.

```
while True:

  print('Who are you?')

  name = input()

  if name != 'Vam':

    continue
  print('Hello, Vam. What is the password? (It is a fish.)')
  password = input()
  if password == 'swordfish':
    break
print('Access granted.')
```

If the user enters any name besides Vam, the continue statement causes the program execution to jump back to the start of the loop. When it reevaluates 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.

FOR LOOPS AND THE RANGE() FUNCTION

The while loop keeps looping while its condition is True (which is the reason for its name), but what if you want to execute a block of code only a certain number of times? You can do this with a for loop statement and the range() function.

```
print('My name is')
for i in range(5):
    print('Python’)
```

```
total = 0
for num in range(101):
   total = total + num
print(total)
```