Q1. Is an assignment operator like += only for show? Is it possible that it would lead to faster results at the runtime?

Operators are used to perform operations on values and variables. These are the special symbols that carry out arithmetic, logical, bitwise computations. The value the operator operates on is known as Operand.

Here, we will cover Assignment Operators in Python. So, Assignment Operators are used to assigning values to variables.

| Operator | Description | Syntax |
| --- | --- | --- |
| -= | Subtract AND: Subtract right operand from left operand and then assign to left operand: True if both operands are equal | a -= b | = | Assign value of right side of expression to left side operand | x = y + z |
| \*= | Multiply AND: Multiply right operand with left operand and then assign to left operand | a \*= b | += | Add and Assign: Add right side operand with left side operand and then assign to left operand | a += b |
| /= | Divide AND: Divide left operand with right operand and then assign to left operand | a /= b |
| %= | Modulus AND: Takes modulus using left and right operands and assign result to left operand | a %= b |
| //= | Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand | a //= b |
| \*\*= | Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand | a \*\*= b |
| &= | Performs Bitwise AND on operands and assign value to left operand | a &= b |
| |= | Performs Bitwise OR on operands and assign value to left operand | a |= b |
| ^= | Performs Bitwise xOR on operands and assign value to left operand | a ^= b |
| >>= | Performs Bitwise right shift on operands and assign value to left operand | a >>= b |
| <<= | Performs Bitwise left shift on operands and assign value to left operand | a <<= b |

Now Let’s see each Assignment Operator one by one.

1) Assign: This operator is used to assign the value of the right side of the expression to the left side operand.

Syntax:

x = y + z

Example:

Python3

|  |
| --- |
| # Assigning values using  # Assignment Operator    a = 3  b = 5    c = a + b    # Output  print(c) |

Output:

8

2) Add and Assign: This operator is used to add the right side operand with the left side operand and then assigning the result to the left operand.

Syntax:

x +=xx

Q2. What is the smallest number of statements you'd have to write in most programming languages to replace the Python expression a, b = a + b, a?

After finishing our previous tutorial on [Python variables](https://realpython.com/python-variables) in this series, you should now have a good grasp of creating and naming Python objects of different types. Let’s do some work with them!

Here’s what you’ll learn in this tutorial: You’ll see how calculations can be performed on objects in Python. By the end of this tutorial, you will be able to create complex expressions by combining objects and operators.

 Take the Quiz: Test your knowledge with our interactive “Python Operators and Expressions” quiz. Upon completion you will receive a score so you can track your learning progress over time:

[Take the Quiz »](https://realpython.com/quizzes/python-operators-expressions/)

In Python, operators are special symbols that designate that some sort of computation should be performed. The values that an operator acts on are called operands.

Here is an example:

>>>

>>> a = 10

>>> b = 20

>>> a + b

30

In this case, the + operator adds the operands a and b together. An operand can be either a literal value or a variable that references an object:

>>>

>>> a = 10

>>> b = 20

>>> a + b - 5

25

A sequence of operands and operators, like a + b - 5, is called an expression. Python supports many operators for combining data objects into expressions. These are explored below.

Arithmetic Operators

The following table lists the arithmetic operators supported by Python:

| Operator | Example | Meaning | Result |
| --- | --- | --- | --- |
| + (unary) | +a | Unary Positive | a In other words, it doesn’t really do anything. It mostly exists for the sake of completeness, to complement Unary Negation. |
| + (binary) | a + b | Addition | Sum of a and b |
| - (unary) | -a | Unary Negation | Value equal to a but opposite in sign |
| - (binary) | a - b | Subtraction | b subtracted from a |
| \* | a \* b | Multiplication | Product of a and b |
| / | a / b | Division | Quotient when a is divided by b. The result always has type float. |
| % | a % b | Modulo | Remainder when a is divided by b |
| // | a // b | Floor Division (also called Integer Division) | Quotient when a is divided by b, rounded to the next smallest whole number |
| \*\* | a \*\* b | Exponentiation | a raised to the power of b |

Here are some examples of these operators in use:

>>>

>>> a = 4

>>> b = 3

>>> +a

4

>>> -b

-3

>>> a + b

7

>>> a - b

1

>>> a \* b

12

>>> a / b

1.3333333333333333

>>> a % b

1

>>> a \*\* b

64

The result of standard division (/) is always a float, even if the dividend is evenly divisible by the divisor:

>>>

>>> 10 / 5

2.0

>>> type(10 / 5)

<class 'float'>

When the result of floor division (//) is positive, it is as though the fractional portion is truncated off, leaving only the integer portion. When the result is negative, the result is rounded down to the next smallest (greater negative) integer:

>>>

>>> 10 / 4

2.5

>>> 10 // 4

2

>>> 10 // -4

-3

>>> -10 // 4

-3

>>> -10 // -4

2

Note, by the way, that in a REPL session, you can display the value of an expression by just typing it in at the >>> prompt without print(), the same as you can with a literal value or variable:

>>>

>>> 25

25

>>> x = 4

>>> y = 6

>>> x

4

>>> y

6

>>> x \* 25 + y

106

[Remove ads](https://realpython.com/account/join/)

Comparison Operators

| Operator | Example | Meaning | Result |
| --- | --- | --- | --- |
| == | a == b | Equal to | True if the value of a is equal to the value of b False otherwise |
| != | a != b | Not equal to | True if a is not equal to b False otherwise |
| < | a < b | Less than | True if a is less than b False otherwise |
| <= | a <= b | Less than or equal to | True if a is less than or equal to b False otherwise |
| > | a > b | Greater than | True if a is greater than b False otherwise |
| >= | a >= b | Greater than or equal to | True if a is greater than or equal to b False otherwise |

Here are examples of the [comparison operators](https://realpython.com/python-is-identity-vs-equality/) in use:

>>>

>>> a = 10

>>> b = 20

>>> a == b

False

>>> a != b

True

>>> a <= b

True

>>> a >= b

False

>>> a = 30

>>> b = 30

>>> a == b

True

>>> a <= b

True

>>> a >= b

True

Comparison operators are typically used in [Boolean](https://realpython.com/python-boolean/) contexts like conditional and loop statements to direct program flow, as you will see later

Q3. In Python, what is the most effective way to set a list of 100 integers to 0?

Python Create List from 0 to 100. A special situation arises if you want to create a list from 0 to 100 (included). In this case, you simply use the list(range(0, 101)) function call.

Python is a very flexible language where a single task can be performed in a number of ways, for example initializing lists can be performed in many ways. However, there are subtle differences in these seemingly similar methods. Python which is popular for its simplicity and readability is equally infamous for being slow compared to C++ or Java. The ‘for’ loop is especially known to be slow whereas methods like map() and filter() are known to be faster because they are written in C. Knowing the better and faster way to initialize lists might give you a slight edge in competitive programming.

The following are some of the ways to initialize lists(we create lists of size 1000 and initialize with zeros) in Python.

Using a for loop and append()  
We create an empty an list and run a for loop for n times using the append() method to add elements to the list.

arr = []

for i in range(100)

arr.append(0)

Using a while loop with a counter variable  
This is similar to the above method. However we use while loop instead.

arr = []

i = 0

while(i<100):

arr.append(0)

Using list comprehensions  
It consists of square brackets containing an expression followed by a for clause and further followed by an optional if clause. The expression can be any type of object that we want to put on the list. Since we are initializing the list with zeros, our expression will just be 0.

arr = [0 for i in range(100)]

Using the \* operator  
The \* operator can be used as [object]\*n where n is the no of elements in the array.

arr = [0]\*100

Lets look at the time taken by each of them. We will calculate the average time taken by each of these methods to initialize an array of 10000 elements over 500 times.

|  |
| --- |
| # import time module to calculate times  import time    # initialize lists to save the times  forLoopTime = []  whileLoopTime = []  listComprehensionTime = []  starOperatorTime = []    # repeat the process for 500 times  # and calculate average of times taken.  for k in range(500):        # start time      start = time.time()      # declare empty list      a = []      # run a for loop for 10000 times      for i in range(100):          a.append(0)      # stop time      stop = time.time()      forLoopTime.append(stop-start)        # start time      start = time.time()      # declare an empty list      a = []      i = 0      # run a for loop 10000 times      while(i<100):          a.append(0)          i+= 1      stop = time.time()      whileLoopTime.append(stop-start)        start = time.time()      # list comprehension to initialize list      a = [0 for i in range(10000)]      stop = time.time()      listComprehensionTime.append(stop-start)          start = time.time()      # using the \* operator      a = [0]\*100      stop = time.time()      starOperatorTime.append(stop-start)        print("Average time taken by for loop: " + str(sum(forLoopTime)/100))  print("Average time taken by while loop: " + str(sum(whileLoopTime)/100))  print("Average time taken by list comprehensions: " + str(sum(listComprehensionTime)/100))  print("Average time taken by \* operator: " + str(sum(starOperatorTime)/100)) |

Output

Average time taken by for loop: 0.012432687282562256

Average time taken by while loop: 0.017907898426055908

Average time taken by list comprehensions: 0.0034629487991333007

Average time taken by \* operator: 0.0001951146125793457

Note: The times will vary based on the platform where this code is executed. These times are just for studying the relative performance of these methods of initializing.

As it can be seen, for and while loops take almost the same time with for loop having a slight edge.

List comprehensions perform much better than for and while loops with the former being around 3-5 times faster. Another example of this difference can be seen when we try to create a list of numbers from 1-1000. Using list comprehensions is much better than using append().

a = [i for i in range(1, 1001)]

Using the \* operator is way faster than the remaining methods and this is the way you should be initializing lists

However one drawback of using the \* operator is while declaring 2d arrays. Using this operator would create shallow lists i.e only one list object would be created and all the indices would refer to this object. This might create unwanted complications. Hence using list comprehensions is a safer way to create 2d lists.

Using \* operator would create shallow lists

arr = [[0]\*no\_of\_cols]\*no\_of\_rows

Using list comprehensions is better for 2d arrays

arr = [[0 for i in range(no\_of\_cols)] for j in range(no\_of\_rows)]

Q4. What is the most effective way to initialise a list of 99 integers that repeats the sequence 1, 2, 3? S If necessary, show step-by-step instructions on how to accomplish this.

The SEQUENCE function in Excel is used to generate an array of sequential numbers such as 1, 2, 3, etc.

It is a new [dynamic array function](https://www.ablebits.com/office-addins-blog/excel-dynamic-arrays-functions-formulas/#dynamic-array-functions) introduced in Microsoft Excel 365. The result is a dynamic array that [spills](https://www.ablebits.com/office-addins-blog/spill-excel-tutorial/) into the specified number of rows and columns automatically.

The function has the following syntax:

SEQUENCE(rows, [columns], [start], [step])

Where:

Rows (optional) - the number of rows to fill.

Columns (optional) - the number of columns to fill. If omitted, defaults to 1 column.

Start (optional) - the starting number in the sequence. If omitted, defaults to 1.

Step (optional) - the increment for each subsequent value in the sequence. It can be positive or negative.

If positive, subsequent values increase, creating an ascending sequence.

If negative, subsequent values decrease, producing a descending sequence.

If omitted, the step defaults to 1.

The SEQUENCE function is only supported in Excel for Microsoft 365, Excel 2021, and Excel for the web.

Basic formula to create a number sequence in Excel

If you are looking to populate a column of rows with sequential numbers starting at 1, you can use the Excel SEQUENCE function in its simplest form:

To put numbers in a column:

SEQUENCE(n)

To place numbers in a row:

SEQUENCE(1, n)

Where n is the number of elements in the sequence.

For example, to populate a column with 10 incremental numbers, type the below formula in the first cell (A2 in our case) and press the Enter key:

=SEQUENCE(10)

The results will spill in the other rows automatically.  
To make a horizontal sequence, set the rows argument to 1 (or omit it) and define the number of columns, 8 in our case:

=SEQUENCE(1,8)  
If you'd like to fill a range of cells with sequential numbers, then define both the rows and columns arguments. For instance, to populate 5 rows and 3 columns, you'd use this formula:

=SEQUENCE(5,3)  
To start with a specific number, say 100, supply that number in the 3rd argument:

=SEQUENCE(5,3,100)  
To generate a list of numbers with a specific increment step, define the step in the 4th argument, 10 in our case:

=SEQUENCE(5,3,100,10)  
Translated into plain English, our complete formula reads as follows:  
SEQUENCE function things to remember

To efficiently do a sequence of numbers in Excel, please remember these 4 simple facts:

The SEQUENCE function is only available with Microsoft 365 subscriptions and Excel 2021. In Excel 2019, Excel 2016 and earlier versions, it does not work since those versions do not support dynamic arrays.

If the array of sequential numbers is the final result, Excel outputs all the numbers automatically in a so-called [spill range](https://www.ablebits.com/office-addins-blog/excel-dynamic-arrays-functions-formulas/#spill-range). So, be sure you have enough empty cells down and to the right of the cell where you enter the formula, otherwise a [#SPILL error](https://www.ablebits.com/office-addins-blog/spill-error-excel-fix/) will occur.

The resulting array can be one-dimensional or two-dimensional, depending on how you configure the rows and columns arguments.

Any optional argument that is not set defaults to 1

Q5. If you're using IDLE to run a Python application, explain how to print a multidimensional list as efficiently?

There can be more than one additional dimension to [lists in Python](https://www.geeksforgeeks.org/python-list/). Keeping in mind that a list can hold other lists, that basic principle can be applied over and over. Multi-dimensional lists are the lists within lists. Usually, a [dictionary](https://www.geeksforgeeks.org/python-dictionary/) will be the better choice rather than a multi-dimensional list in Python.

Accessing a multidimensional list:

Approach 1:

|  |
| --- |
| # Python program to demonstrate printing  # of complete multidimensional list  a = [[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]  print(a) |

Output:

[[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]

Q6. Is it possible to use list comprehension with a string? If so, how can you go about doing it?

List comprehension in Python is an easy and compact syntax for creating a list from a string or another list. It is a very concise way to create a new list by performing an operation on each item in the existing list. List comprehension is considerably faster than processing a list using the for loop.

List Comprehension Syntax:

[expression for element in iterable if condition]

As per the above syntax, the list comprehension syntax contains three parts: an expression, one or more [for loop](https://www.tutorialsteacher.com/python/python-for-loop), and optionally, one or more [if conditions](https://www.tutorialsteacher.com/python/python-if-elif). The list comprehension must be in the square brackets []. The result of the first expression will be stored in the new list. The for loop is used to iterate over the iterable object that optionally includes the if condition.

Suppose we want to find even numbers from 0 to 20 then we can do it using a [for loop](https://www.tutorialsteacher.com/python/python-for-loop), as shown below:

Example: Create List of Even Numbers without List Comprehension

 Copy

even\_nums = []

for x in range(21):

if x%2 == 0:

even\_nums.append(x)

print(even\_nums)

Output

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

The same result can be easily achieved using a list comprehension technique shown below.

Example: Create List of Even Numbers with List Comprehension

 Copy

even\_nums = [x for x in range(21) if x%2 == 0]

print(even\_nums)

Output

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

In the above example, [x for x in range(21) if x%2 == 0] returns a new list using the list comprehension. First, it executes the for loop for x in range(21) if x%2 == 0. The element x would be returned if the specified condition if x%2 == 0 evaluates to True. If the condition evaluates to True, then the expression before for loop would be executed and stored in the new list. Here, expression x simply stores the value of x into a new list.

Q7. From the command line, how do you get support with a user-written Python programme? Is this possible from inside IDLE?

Starting IDLE on Mac

In a Terminal window, type python. This will start the Python shell. The prompt for that is >>>

At the Python shell prompt type import idlelib.idle.

This will start the IDLE IDE.

Getting Started with Python Programming for Mac Users

Python comes bundled with Mac OS X. But the version that you have is quite likely an older version. [Download](http://www.python.org/download/mac/)the latest binary version of Python that runs on both Power PC and Intel systems and install it on your system.

Writing Your First Python Program

Click on File and then New Finder Window.

Click on Documents.

Click on File and then New Folder.

Call the folder CS313E. You will be storing all class related programs there.

Click on Applications and then TextEdit.

Click on TextEdit on the menu bar and select Preferences.

Select Plain Text.

In the empty TextEdit window type in the following program, exactly as given:

# File: Hello.py

print "Hello World!"

From the File menu in TextEdit click on Save As.

In the field Save As: type Hello.py.

Select Documents and the file folder CS313E.

Click Save.

Running Your First Program

Select Applications, then Utilities and Terminal.

In your Terminal window type ls and Return. It should give a listing of all the top level folders. You should see the Documents folder.

Type cd Documents and hit Return.

Type ls and hit Return and you should see the folder CS313E.

Type cd CS313E and hit Return.

Type ls and hit return and you should see the file Hello.py.

To run the program, type python Hello.py and hit Return.

You should see the line Hello World!

Congratulations, you have run your first Python program.

Starting IDLE on Mac

In a Terminal window, type python. This will start the Python shell. The prompt for that is >>>

At the Python shell prompt type import idlelib.idle

This will start the IDLE IDE

Using IDLE on either Windows or Mac

Start IDLE

Go to File menu and click on New Window

Type your program in

Go to File menu and click on Save. Type in filename.py This will save it as a plain text file, which can be opened in in any editor you choose (like Notepad or TextEdit).

To run your program go to Run and click Run Module

Q8. Functions are said to be “first-class objects” in Python but not in most other languages, such as C++ or Java. What can you do in Python with a function (callable object) that you can't do in C or C++?

C++ is faster than Python because it is statically typed, which leads to a faster compilation of code. Python is slower than C++, it supports dynamic typing, and it also uses the interpreter, which makes the process of compilation slower.

[Python](https://www.simplilearn.com/why-learn-python-a-guide-to-unlock-your-python-career-article) and [C++](https://www.simplilearn.com/tutorials/cpp-tutorial/learn-cpp-basics) are both general-purpose [programming languages](https://www.simplilearn.com/best-programming-languages-start-learning-today-article). Both languages are pretty different when it comes to overall approaches like syntax, usage, etc. But still, it gets difficult while you select the correct language. In this tutorial on C++ vs. Python, you will cover everything you need to know about C++ and Python so that it gets easy for you to select a suitable language

What Is C++?

C++ is a programming language that Bjarne Stroustrup introduced in 1979 in New Jersey. It is a general-purpose programming language that means it is used to build software for a wide variety of applications. Earlier it was called C with classes instead of C++ because it was created as an extension of the [C language.](https://www.simplilearn.com/c-programming-article)

C++ is considered as an intermediate-level programming language, as it contains both the features of high-level programming and low-level programming language.

Syntax of C++

The syntax of C++ includes.

Header files

Main function

Class

Methods

Objects

Instance variables

Return statement

Header files: Header files that are included at the top of the program. These files instruct the compiler to include all the required functions associated with the header file. This is included using the preprocessor directive #include.

The <iostream> is the header file that is generally included in the C++ programs, and this file allows us to use input-output operations.

Main function: The main function can be called an entry point from where the execution of our program starts. Every C++ program contains the main function, and whenever the program is executed, the control directly goes to the main function.

Class: A class can be defined as a blueprint that describes the behavior of objects, or you can say it is a collection of objects. Class is defined with the keyword class, and it has its own members called member functions or methods, variables and constructors, etc.

Methods: Method or function is one of the most important parts of C++ programming because it is used to write logic, perform data manipulation, and all other essential operations inside the method. You can give the name to the function along with the return type, and then inside the brackets, we can write the operations or logic.

Objects: Objects can be defined as an instance of a class. If an object is created for a particular class, then that object can be used to access data members and functions of that class using the dot operator. It can be done by writing the name of the object preceding the dot operator and then the function name following round brackets.

Instance variables: These are those variables that are defined inside the class but are outside the methods. They basically belong to the object, and every object has its own unique set of instance variables.

|  |  |
| --- | --- |
| C++ | Python |
| Simplicity of language  C++ is a bit complex when it comes to the simplicity of language, and it has more syntax rules as well as program conventions. | Simplicity of language  Python is a friendly language. It has a simple and easy-to-learn syntax. Moreover, its features are easy to use, which allows you to write short and readable code. |
| Speed  C++ is faster than Python because it is statically typed, which leads to a faster compilation of code. | Speed  Python is slower than C++, it supports dynamic typing, and it also uses the interpreter, which makes the process of compilation slower. |
| Memory management  In C++, memory management takes place manually as it doesn’t have any garbage collector. Moreover, it uses pointers which make it more vulnerable to memory leaks. | Memory management  Python provides automatically programmed memory management as there is a garbage collector in python. |
| Declaration  If you want to declare any variable, then you need to declare it with its type. | Declaration  In Python, you don’t need to declare any data type with the variable. |
| Usage  Python is widely used in technologies such as machine learning, artificial intelligence, etc., and popularly used in back-end development. | Usage  C++ is good in hardware-level coding; that is why it is widely used to make embedded systems. |

Q9. How do you distinguish between a wrapper, a wrapped feature, and a decorator?

Wrappers around the functions are also knows as decorators which are a very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of the wrapped function, without permanently modifying it.

Wrappers around the functions are also knows as [decorators](https://www.geeksforgeeks.org/decorators-in-python/) which are a very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of the wrapped function, without permanently modifying it. In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.

Syntax:

@wrapper

def function(n):

statements(s)

This is also similar to

def function(n):

statement(s)

function = wrapper(function

Differences

Intention:  The end product may look similar but the intention is different. A wrapper as used in the Facade pattern is intended to simplify an interface to an external library. An adapter is intended to bridge the disconnect between one interface and another. You may look at a new library that you wish to use and write a wrapper to simplify and streamline its use. You may look at an interface, internal or external, that your existing code needs to conform to, and write an adapter to do that.

Composition:  A wrapper contains another object and wraps around it. It has the the sole responsibility of moving data to and from the wrapped object. An adapter doesn’t necessarily contain or simplify an object, although this can be a secondary benefit of using adapters. An adapter transforms input to make it match the input required by another interface. It adapts input to that other interface.

Problem space:  An adapter solves a problem of incompatibility, while a wrapper fulfills the need of a simplified and specific programming interface.

Q10. If a function is a generator function, what does it return?

Python provides a generator to create your own [iterator function](https://www.tutorialsteacher.com/python/iter-method). A generator is a special type of function which does not return a single value, instead, it returns an iterator object with a sequence of values. In a generator function, a yield statement is used rather than a return statement. The following is a simple generator function.

Example: Generator Function

 Copy

def mygenerator():

print('First item')

yield 10

print('Second item')

yield 20

print('Last item')

yield 30

In the above example, the mygenerator() function is a generator function. It uses yield instead of return keyword. So, this will return the value against the yield keyword each time it is called. However, you need to create an iterator for this function, as shown below.

Example: next()

 Copy

>>> gen = mygenerator()

>>> next(gen)

First item

10

>>> next(gen)

Second item

20

>>> next(gen)

Last item

30

The generator function cannot include the return keyword. If you include it, then it will terminate the function. The difference between yield and return is that yield returns a value and pauses the execution while maintaining the internal states, whereas the return statement returns a value and terminates the execution of the function.

Q11. What is the one improvement that must be made to a function in order for it to become a generator function in the Python language?

It is fairly simple to create a generator in Python. It is as easy as defining a normal function, but with a yield statement instead of a return statement. If a function contains at least one yield statement (it may contain other yield or return statements), it becomes a generator function.

Differences between Generator function and Normal function

Here is how a generator function differs from a normal [function](https://www.programiz.com/python-programming/function).

Generator function contains one or more yield statements.

When called, it returns an object (iterator) but does not start execution immediately.

Methods like \_\_iter\_\_() and \_\_next\_\_() are implemented automatically. So we can iterate through the items using next().

Once the function yields, the function is paused and the control is transferred to the caller.

Local variables and their states are remembered between successive calls.

Finally, when the function terminates, StopIteration is raised automatically on further calls.

Here is an example to illustrate all of the points stated above. We have a generator function named my\_gen() with several yield statements.

# A simple generator function

def my\_gen():

n = 1

print('This is printed first')

# Generator function contains yield statements

yield n

n += 1

print('This is printed second')

yield n

n += 1

print('This is printed at last')

yield n

An interactive run in the interpreter is given below. Run these in the Python shell to see the output.

>>> # It returns an object but does not start execution immediately.

>>> a = my\_gen()

>>> # We can iterate through the items using next().

>>> next(a)

This is printed first

1

>>> # Once the function yields, the function is paused and the control is transferred to the caller.

>>> # Local variables and theirs states are remembered between successive calls.

>>> next(a)

This is printed second

2

>>> next(a)

This is printed at last

3

>>> # Finally, when the function terminates, StopIteration is raised automatically on further calls.

>>> next(a)

Traceback (most recent call last):

...

StopIteration

>>> next(a)

Traceback (most recent call last):

...

StopIteration

One interesting thing to note in the above example is that the value of variable n is remembered between each call.

Unlike normal functions, the local variables are not destroyed when the function yields. Furthermore, the generator object can be iterated only once.

To restart the process we need to create another generator object using something like a = my\_gen().

One final thing to note is that we can use generators with [for loops](https://www.programiz.com/python-programming/for-loop) directly.

This is because a for loop takes an iterator and iterates over it using next() function. It automatically ends when StopIteration is raised. Check here to [know how a for loop is actually implemented in Python](https://www.programiz.com/python-programming/iterator#for-loop-working).

# A simple generator function

def my\_gen():

n = 1

print('This is printed first')

# Generator function contains yield statements

yield n

n += 1

print('This is printed second')

yield n

n += 1

print('This is printed at last')

yield n

# Using for loop

for item in my\_gen():

print(item)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

When you run the program, the output will be:

This is printed first

1

This is printed second

2

This is printed at last

3

Q12. Identify at least one benefit of generators

Emergency Power. ...

Power for Appliances. ...

Power for Tools. ...

Recreational Uses. ...

Damage Prevention. ...

Suitability. ...

Safe and Easy to Use. ...

Rugged and Dependable