Python Programming Guide

INDEX

STANDARD INPUT **3**

Input 3

STANDARD OUTPUT **3**

Print 3

SIMPLE MATH **4**

[+, -, *, /] Operators 4

FLOW CONTROL **5**

If 5
Switch 5
For 6
While 6

STRING MANIPULATION **7**

String manipulation 7
Split 7
ASCII lowercase-uppercase conversion 8

DECIMALS **9**

Decimal rounding 9
Decimal approximation 9
Setting number of decimals to print 10
Float, Double 10

SIMPLE DATA STRUCTURES **11**

Array (list) 11
Matrix 11
Dictionary 12

BASIC ALGORITHMS **13**

Sort 13
Search 13

COMPLEX MATH **14**

Modulus 14
Square root 14
Power 15
Trigonometry 15

COMPLEX DATA STRUCTURES **16**

Dynamic array 16
Stack 16
Queue 17

STANDARD INPUT

Input

In Python, the input() function is used to receive input from the keyboard. In this case, it's used to receive the user's name. Then, the text "Hello " is concatenated with the user's name and printed to the console.

SOURCE CODE	INPUT	OUTPUT
name = input() print("Hello " + name)	Maria	Hello Maria

STANDARD OUTPUT

Print

In Python, the print() function is used to output text to the console. In this case, the text "Hello World" is printed to the console.

SOURCE CODE	OUTPUT
print("Hello World")	Hello World

SIMPLE MATH

[+, -, *, /] Operators

In Python, you can perform basic mathematical operations such as addition, subtraction, multiplication, and division using the +, -, *, and / operators respectively. Here is an example code snippet that demonstrates simple math operations.

SOURCE CODE	OUTPUT
# Addition a = 5 b = 10 c = a + b print(c) # Output: 15 **# Subtraction** a = 7 b = 2 c = a - b print(c) # Output: 5 **# Multiplication** a = 3 b = 4 c = a * b print(c) # Output: 12 **# Division** a = 10 b = 3 c = a / b print(c) # Output: 3.33333333333	15 5 12 3.33333333333

FLOW CONTROL

If

In Python, the if statement is used for conditional execution. In this example, the value of num is checked to determine whether it is positive, zero, or negative using if, elif, and else statements. The appropriate message is printed to the console based on the condition.

SOURCE CODE	OUTPUT

num = 10
if num > 0:
  print("Positive number")
elif num == 0:
  print("Zero")
else:
  print("Negative number")
	Positive number

Switch

In Python, the switch statement does not exist natively. However, you can use a dictionary to achieve similar functionality. In this example, a function called num_to_string takes a number as input and returns a string based on the value of the number using a dictionary. The appropriate message is returned based on the condition.

SOURCE CODE	OUTPUT

def num_to_string(num):
  switcher = {
  1: "One",
  2: "Two",
  3: "Three",
  4: "Four",
  5: "Five"
  }
  return switcher.get(num, "Invalid number")
 	
print(num_to_string(2))
	Two

For

In Python, the for loop iterates over a sequence of values. In this example, the range function generates a sequence of numbers from 1 to 4, and the loop prints each value in the sequence to the console.

SOURCE CODE	OUTPUT
for i in range(1, 5): print(i)	1 2 3 4

While

In Python, the while loop executes a block of code while a condition is true. In this example, the loop initializes a counter variable i to 1, and continues to print the value of i to the console and increment i by 1 until i is no longer less than or equal to 4.

SOURCE CODE	OUTPUT

i = 1
while i <= 4:
  print(i)
  i += 1
	1
	2
	3
	4

STRING MANIPULATION

String manipulation

Python provides a rich set of functions for string manipulation. You can manipulate strings in a variety of ways such as slicing, concatenating, formatting, and replacing.

Here's an example code snippet that demonstrates some string manipulation operations:

SOURCE CODE	OUPUT
# String slicing s = "Hello, World!" print(s[1:5]) # Output: ello **# String concatenation** s1 = "Hello, " s2 = "World!" s = s1 + s2 print(s) # Output: Hello, World! **# String formatting** name = "John" age = 30 s = "My name is {} and I'm {} years old".format(name, age) print(s) # Output: My name is John and I'm 30 years old **# String replacing** s = "Hello, World!" s = s.replace("World", "Python") print(s) # Output: Hello, Python!	ello Hello, World! My name is John and I'm 30 years old Hello, Python!

Split

In Python, the split() function is used to split a string into a list of substrings based on a delimiter. By default, the delimiter is a space character. Here's an example code snippet that demonstrates the use of the split() function.

SOURCE CODE	OUTPUT
s = "The quick brown fox jumps over the lazy dog" words = s.split() print(words)	['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

ASCII lowercase-uppercase conversion

ASCII values for lowercase characters range from 97 to 122, while ASCII values for uppercase characters range from 65 to 90. To convert a lowercase character to uppercase in ASCII, we can add the difference between the ASCII value of 'a' and 'A' to the ASCII value of the lowercase character. Here's an example code snippet:

SOURCE CODE	OUTPUT

# create a string 	
my_str = "hello world"
 	
# create an empty string to hold the manipulated string 	
manipulated_str = ""
 	
# loop over each character in the string 	
for c in my_str:
  # check if the character is lowercase
  if c.islower():
  # convert lowercase to uppercase by adding the ASCII difference
  manipulated_str += chr(ord(c) - 32)
  else:
  # leave the character unchanged
  manipulated_str += c
 	
print(manipulated_str)
	HELLO WORLD

DECIMALS

Decimal rounding

In Python, you can use the round() function to round a decimal number to a specified number of digits. By default, round() rounds to the nearest integer. However, you can specify the number of digits to which to round. Here's an example code snippet that demonstrates the use of the round() function.

SOURCE CODE	OUTPUT
x = 3.14159265359 print(round(x, 2))	3.14

Decimal approximation

In Python, the // operator is used to perform floor division. This operator divides two numbers and returns the largest integer that is less than or equal to the result. Here's an example code snippet that demonstrates floor division.

SOURCE CODE	OUTPUT
a = 10 b = 3 print(a // b)	3

To perform ceiling division in Python, you can use the math.ceil() function, which returns the smallest integer that is greater than or equal to the result. Here's an example code snippet that demonstrates ceiling division.

SOURCE CODE	OUTPUT
import math a = 10 b = 3 print(math.ceil(a / b))	4

Setting number of decimals to print

In Python, you can set the number of decimal places to print using the round() function, as shown in the first example. Alternatively, you can use string formatting to print floating point numbers with a specified number of decimal places, as shown in the second example.

SOURCE CODE	OUTPUT
# Setting the number of decimal places to print x = 3.14159 rounded_x = round(x, 2) print(rounded_x) **# Another way to format floating point numbers with a specified number of decimal places** y = 2.71828 print("{:.2f}".format(y))	3.14 2.72

Float, Double

In Python, float and double are both represented by the float data type. Here's an example code snippet in Python to demonstrate how to manipulate float and double values:

SOURCE CODE	OUTPUT
# create a float my_float = 1.23456789 **# create a double** my_double = 1.234567890123456789 **# manipulate the values** manipulated_float = my_float * 2 manipulated_double = my_double * 2 print("Float: {:.10f}".format(manipulated_float)) print("Double: {:.20f}".format(manipulated_double))	Float: 2.4691357800 Double: 2.46913578024691338086

SIMPLE DATA STRUCTURES

Array (list)

In Python, a list can be used to represent an array. In this example, a numbers list is declared with 5 integer values. Then, the entire list is printed to the console using print().

SOURCE CODE	OUTPUT
numbers = [1, 2, 3, 4, 5] print(numbers)	[1, 2, 3, 4, 5]

Matrix

In Python, matrices can be represented using nested lists, or using the numpy module for more advanced matrix operations. Here's an example code snippet in Python to demonstrate how to manipulate matrices using nested lists:

SOURCE CODE	OUTPUT

# create a matrix 	
my_matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
 	
# manipulate the values 	
for i in range(len(my_matrix)):
  for j in range(len(my_matrix[0])):
  my_matrix[i][j] *= 2
 	
print("Matrix: ")
for row in my_matrix:
  print(row)
	Matrix:
	[2, 4, 6]
	[8, 10, 12]
	[14, 16, 18]

Dictionary

In Python, dictionaries are implemented as an unordered collection of key-value pairs, where each key must be unique. The keys in a dictionary are used to retrieve the corresponding values.

SOURCE CODE	OUTPUT
# Creating a dictionary my_dict = {'apple': 2, 'banana': 4, 'orange': 6} **# Accessing values in a dictionary** print(my_dict['apple']) # prints 2 print(my_dict['orange']) # prints 6 **# Adding key-value pairs to a dictionary** my_dict['grape'] = 8 print(my_dict) # prints {'apple': 2, 'banana': 4, 'orange': 6, 'grape': 8} **# Removing key-value pairs from a dictionary** del my_dict['banana'] print(my_dict) # prints {'apple': 2, 'orange': 6, 'grape': 8}	2 6 {'apple': 2, 'banana': 4, 'orange': 6, 'grape': 8} {'apple': 2, 'orange': 6, 'grape': 8}

BASIC ALGORITHMS

Sort

In Python, sorting can be performed on arrays using the built-in sorted() function, or by using the sort() method of the array object. Here's an example code snippet in Python to demonstrate how to sort an array:

SOURCE CODE	OUTPUT
# create an array my_array = [5, 2, 8, 1, 9] **# sort the array** sorted_array = sorted(my_array) **# print the sorted array** print("Sorted array: ", sorted_array) **# sort the array in-place** my_array.sort() **# print the sorted array** print("Sorted array (in-place): ", my_array)	Sorted array: [1, 2, 5, 8, 9] Sorted array (in-place): [1, 2, 5, 8, 9]

Search

In Python, searching can be performed on arrays using the built-in in operator, the index() method of the array object, or by iterating through the array. Here's an example code snippet in Python to demonstrate how to search an array:

SOURCE CODE	OUTPUT
# create an array my_array = [5, 2, 8, 1, 9] **# check if a value is in the array** print("5 is in the array: ", 5 in my_array) **# find the index of a value in the array** print("Index of 8: ", my_array.index(8)) **# find all indices of a value in the array** indices = [i for i in range(len(my_array)) if my_array[i] == 8] print("Indices of 8: ", indices)	5 is in the array: True Index of 8: 2 Indices of 8: [2]

COMPLEX MATH

Modulus

In Python, the modulus operation can be performed using the % operator. The modulo operation returns the remainder when one number is divided by another. Here's an example code snippet in Python to demonstrate the modulo operation:

SOURCE CODE	OUTPUT
# modulus operation num1 = 10 num2 = 7 result = num1 % num2 print("Modulus: ", result)	Modulus: 3

Square root

In Python, the square root operation can be performed using the math module. The math.sqrt() function returns the square root of a number. Here's an example code snippet in Python to demonstrate the square root operation:

SOURCE CODE	OUTPUT
import math **# square root operation** num = 16 result = math.sqrt(num) print("Square root: ", result)	Square root: 4.0

Power

In Python, the power operation can be performed using the ** operator or the math.pow() function. The ** operator raises the first operand to the power of the second operand. The math.pow() function raises the first operand to the power of the second operand and returns the result as a float. Here's an example code snippet in Python to demonstrate the power operation:

SOURCE CODE	OUTPUT
import math **# power operation** num = 3 exponent = 2 result1 = num ** exponent result2 = math.pow(num, exponent) print("Power using ** operator: ", result1) print("Power using math.pow(): ", result2)	Power using ** operator: 9 Power using math.pow(): 9.0

Trigonometry

In Python, trigonometric operations can be performed using the math module. The math.sin(), math.cos(), and math.tan() functions return the sine, cosine, and tangent of an angle in radians, respectively. Here's an example code snippet in Python to demonstrate trigonometric operations:

SOURCE CODE	OUTPUT
import math **# trigonometric operations** theta = math.pi / 4 sine = math.sin(theta) cosine = math.cos(theta) tangent = math.tan(theta) print("Sine: ", sine) print("Cosine: ", cosine) print("Tangent: ", tangent)	Sine: 0.7071067811865475 Cosine: 0.7071067811865476 Tangent: 0.9999999999999999

COMPLEX DATA STRUCTURES

Dynamic array

In Python, dynamic arrays can be created using the built-in list type. Elements can be added to the dynamic array using the append() method. The dynamic array can be printed using the print() function.

SOURCE CODE	OUTPUT
# Creating a dynamic array in Python using the built-in list type my_list = [] my_list.append(1) my_list.append(2) my_list.append(3) print(my_list)	[1, 2, 3]

Stack

In Python, you can implement a stack lists. For a stack, you can use the append() method to add elements to the end of the list and the pop() method to remove the last element from the list.

SURCE CODE	OUTPUT
# Creating a stack my_stack = [] **# Adding elements to the stack** my_stack.append(1) my_stack.append(2) my_stack.append(3) **# Removing elements from the stack** print(my_stack.pop()) print(my_stack.pop()) print(my_stack.pop())	3 2 1

Queue

In Python, you can implement a queue using lists. For a queue, you can use the append() method to add elements to the end of the list and the pop(0) method to remove the first element from the list.

SOURCE CODE	OUTPUT
# Creating a queue my_queue = [] **# Adding elements to the queue** my_queue.append(1) my_queue.append(2) my_queue.append(3) **# Removing elements from the queue** print(my_queue.pop(0)) # prints 1 print(my_queue.pop(0)) # prints 2 print(my_queue.pop(0)) # prints 3	1 2 3