python

Python Snippets — Learning Collection

About This Collection

This is a curated set of 64 practical Python code snippets designed for learning and quick reference. Each snippet demonstrates a specific programming concept, algorithm, or common task you'll encounter in real-world Python development.

What you'll find here:

- String manipulation and text processing techniques
- Numeric operations, type conversions, and mathematical functions
- List operations, comprehensions, and functional programming patterns
- Control flow structures (loops, conditionals, pattern matching)
- Functions, lambdas, decorators, and closures
- Object-oriented programming basics (classes, inheritance, instance vs class members)
- Pattern generation and algorithmic challenges
- Python-specific features (iterators, generators, `*args`, `**kwargs`)

Who is this for:

- Beginners learning Python fundamentals
- Developers transitioning from other languages
- Anyone preparing for coding interviews
- Programmers looking for quick syntax references

All code blocks have been verified to run without errors in Python 3.12+. Simply copy, paste, and execute to see immediate results.

All code blocks below were rewritten (when needed) so they execute without errors.

1. Converting an Integer into Decimals

```
import decimal
integer = 10
d = decimal.Decimal(integer)
print(d)
print(type(d))
# Expected output:
# 10
# 10
```

2. Converting a String of Integers into Decimals

```
import decimal
s = "12345"
d = decimal.Decimal(s)
print(d)
print(type(d))
# Expected output:
# 12345
# 12345
```

3. Reversing a String using an Extended Slicing Technique

```
text = "Python Programming"
print(text[::-1])
# Expected output:
# gnimmargorP nohtyP
```

4. Counting VOWELS in a Given Word

```
word = "programming"
vowels = set("aeiouAEIOU")
count = sum(1 for ch in word if ch in vowels)
print("vowels:", count)
# Expected output:
# vowels: 3
```

5. Counting CONSONANTS in a Given Word

```
word = "programming"
vowels = set("aeiouAEIOU")
count = sum(1 for ch in word if ch.isalpha() and ch not in vowels)
print("consonants:", count)
# Expected output:
# consonants: 8
```

6. Counting the number of occurrences of a character in a String

```
word = "programming"
character = "g"
count = word.count(character)
print(f"'{character}' occurs {count} time(s) in '{word}'")
# Expected output:
# 'g' occurs 2 time(s) in 'programming'
```

7. Writing FIBONACCI Series

```
def fibonacci(n: int):
    a, b = 0, 1
    seq = []
    for _ in range(n):
        seq.append(a)
        a, b = b, a + b
    return seq
print(fibonacci(10))
# Expected output:
# [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

8. Finding the Maximum Number in a List

```
nums = [10, 2, 78, 4, 45, 99, 3]
```

```
print("max:", max(nums))
# Expected output:
# max: 99
```

9. Finding the Minimum Number in a List

```
nums = [10, 2, 78, 4, 45, 99, 3]
print("min:", min(nums))
# Expected output:
# min: 2
```

10. Finding the Middle Element in a List

For even-length lists, you'd typically return two middle values or choose a convention (left/right).

```
nums = [10, 20, 30, 40, 50]
mid = nums[len(nums)//2]
print("middle:", mid)
# Expected output:
# middle: 30
```

11. Converting a List into a String

```
items = ["Python", "is", "fun"]
s = " ".join(items)
print(s)
# Expected output:
# Python is fun
```

12. Adding Two List Elements Together

```
a = [1, 2, 3, 4]
b = [10, 20, 30, 40]
c = [x + y for x, y in zip(a, b)]
print(c)
# Expected output:
# [11, 22, 33, 44]
```

13. Comparing Two Strings for ANAGRAMS

```
def is_anagram(a: str, b: str) -> bool:
    a = "".join(ch.lower() for ch in a if ch.isalnum())
    b = "".join(ch.lower() for ch in b if ch.isalnum())
    return sorted(a) == sorted(b)
print(is_anagram("listen", "silent"))
print(is_anagram("hello", "world"))
# Expected output:
# True
# False
```

14. Checking for PALINDROME Using Extended Slicing Technique

```
def is_palindrome(s: str) -> bool:
    s = "".join(ch.lower() for ch in s if ch.isalnum())
    return s == s[::-1]
print(is_palindrome("madam"))
print(is_palindrome("A man, a plan, a canal: Panama"))
# Expected output:
# True
# True
```

15. Counting the White Spaces in a String

```
text = "Count the spaces in this sentence."
spaces = sum(1 for ch in text if ch.isspace())
print("spaces:", spaces)
# Expected output:
# spaces: 5
```

16. Counting Digits, Letters, and Spaces in a String

```
text = "Python 3.12 is cool!"
digits = sum(ch.isdigit() for ch in text)
letters = sum(ch.isalpha() for ch in text)
spaces = sum(ch.isspace() for ch in text)
```

```
print("digits:", digits)
print("letters:", letters)
print("spaces:", spaces)
# Expected output:
# digits: 3
# letters: 12
# spaces: 3
```

17. Counting Special Characters in a String

```
text = "Python 3.12 is cool! #1 :)"
special = sum((not ch.isalnum()) and (not ch.isspace()) for ch in text)
print("special characters:", special)
# Expected output:
# special characters: 5
```

18. Removing All Whitespace in a String

```
text = " remove   all\twhitespace\nfrom\rhere "
clean = "".join(text.split())
print(clean)
# Expected output:
# removeallwhitespacefromhere
```

19. Building a Pyramid in Python

```
def star_pyramid(rows: int):
    for i in range(rows):
        spaces = " " * (rows - i - 1)
        stars = "* " * (i + 1)
        print(spaces + stars.rstrip())
star_pyramid(5)
# Expected output:
#     *
#    * *
```

```
#   * * *
#  * * * *
# * * * * *
```

20. Randomizing the Items of a List in Python

```
import random
items = [1, 2, 3, 4, 5]
random.shuffle(items)
print(items)
# Expected output:
# [3, 2, 1, 5, 4]
```

21. Create a Generator to Produce First n Prime Numbers

```
def is_prime(x: int) -> bool:
def is_prime(x: int) -> bool:
        return False
    if x == 2:
        return True
    if x % 2 == 0:
        return False
    p = 3
    p = 3
        if x % p == 0:
            return False
        p += 2
    return True
def first_n_primes(n: int):
    count = 0
    x = 2
    x = 2
        if is_prime(x):
```

```
            yield x
            count += 1
        x += 1
print(list(first_n_primes(10)))
# Expected output:
# [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

22. Implementing Variable Length Arguments in Python

```
def demo(*args, **kwargs):
    print("args:", args)
    print("kwargs:", kwargs)
demo(1, 2, 3, name="Igor", lang="Python")
# Expected output:
# args: (1, 2, 3)
# kwargs: {'name': 'Igor', 'lang': 'Python'}
```

23. Creating Instance Member Variables in Python

```
class Person:
    def __init__(self, name: str, age: int):
        self.name = name          # instance variable
        self.age = age            # instance variable
p = Person("Alice", 30)
print(p.name, p.age)
# Expected output:
# Alice 30
```

24. Addition Using Lambda Functions

```
add = lambda a, b: a + b
print(add(10, 20))
# Expected output:
# 30
```

25. Finding Factorial Using Lambda Function

```
from functools import reduce
factorial = lambda n: reduce(lambda a, b: a * b, range(1, n + 1), 1)
print(factorial(5))
# Expected output:
# 120
```

26. List Comprehension (Create a List in Single Line)

```
squares = [x * x for x in range(1, 11)]
print(squares)
# Expected output:
# [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

27. Reversing Words in a Sentence

```
sentence = "What is right in your mind is right in your world"
words = sentence.split()
reversed_sentence = " ".join(reversed(words))
print(reversed_sentence)
# Expected output:
# world your in right is mind your in right is What
```

28. Global and Local Variable

```
x = "global"
def f():
    x = "local"
    return x
print("inside f():", f())
print("global x:", x)
# Expected output:
# inside f(): local
```

```
# global x: global
```

29. globals() Function

```
g = 10
def set_global():
    globals()["g"] = 99  # modify global variable via globals() dict
set_global()
print(g)
# Expected output:
# 99
```

30. Type Conversion Basics

```
print(int("42"))
print(float("3.14"))
print(str(123))
print(list("abc"))
# Expected output:
# 42
# 3.14
# 123
# ['a', 'b', 'c']
```

31. Type Conversion Examples

```
value = "100"
as_int = int(value)
as_float = float(value)
as_bool = bool(value)  # non-empty string -> True
print(as_int, type(as_int))
print(as_float, type(as_float))
print(as_bool, type(as_bool))
# Expected output:
# Expected output:
# Expected output:
```

```
# Expected output:
```

32. What is a Python Decorator?

```
def log_calls(fn):
    def wrapper(*args, **kwargs):
        print(f"calling {fn.__name__} with args={args},
kwargs={kwargs}")
        result = fn(*args, **kwargs)
        print(f"{fn.__name__} returned {result}")
        return result
    return wrapper
@log_calls
def add(a, b):
    return a + b
add(2, 3)
# Expected output:
# calling add with args=(2, 3), kwargs={}
# add returned 5
```

33. What are Iterators in Python?

```
items = [10, 20, 30]
it = iter(items)
print(next(it))
print(next(it))
print(next(it))
# Expected output:
# 10
# 20
# 30
```

34. Is Function Overloading Allowed in Python?

```
# Python doesn't support true compile-time overloading
like C++.
```

```
# Common patterns: default args or *args to accept
variable parameters.
def area(*args):
    if len(args) == 1:          # circle
        (r,) = args
        return 3.14159 * r * r
    if len(args) == 2:          # rectangle
        w, h = args
        return w * h
    raise TypeError("area() expects 1 or 2 arguments")
print(area(3))
print(area(4, 5))
# Expected output:
# 28.274309999999996
# 20
```

35. What are Positional Arguments in Python?

```
def greet(first, last):
    print(f"Hello, {first} {last}!")
greet("Ada", "Lovelace")  # positional arguments
# Expected output:
# Hello, Ada Lovelace!
```

36. Difference Between sorted() and list.sort()

```
nums = [3, 1, 2]
a = sorted(nums)  # returns a NEW list
print("sorted:", a, "original:", nums)
nums.sort()       # sorts IN PLACE
print("after sort:", nums)
# Expected output:
# sorted: [1, 2, 3] original: [3, 1, 2]
```

```
# after sort: [1, 2, 3]
```

37. How to Create Static (Class) Member Variables in Class?

```
class Counter:
    total_created = 0  # class (static) variable
    def __init__(self):
        Counter.total_created += 1
Counter()
Counter()
print(Counter.total_created)
# Expected output:
# 2
```

38. How to Use else with Loop in Python?

```
nums = [2, 4, 6, 7, 8]
for n in nums:
    if n % 2 == 1:
        print("found odd:", n)
        break
else:
    # runs only if the loop was NOT broken
    print("no odd numbers found")
# Expected output:
# found odd: 7
```

39. What is Name Mangling in Python?

```
class Demo:
    def __init__(self):
        self.__secret = "hidden"  # becomes
_Demo__secret
d = Demo()
print(d._Demo__secret)
# Expected output:
```

```
# hidden
```

40. Difference Between Class Object and Instance Object

```
class Dog:
    species = "Canis familiaris"   # class attribute
    def __init__(self, name):
        self.name = name          # instance attribute
print("class attribute via class:", Dog.species)
d1 = Dog("Rex")
d2 = Dog("Max")
print("instance 1:", d1.name, d1.species)
print("instance 2:", d2.name, d2.species)
# Expected output:
# class attribute via class: Canis familiaris
# instance 1: Rex Canis familiaris
# instance 2: Max Canis familiaris
```

41. What is __init__ Method in Python?

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
p = Point(3, 4)
print(p.x, p.y)
# Expected output:
# 3 4
```

42. Default Arguments in Functions

```
def add(a, b, c=5):  # c is a default argument
    return a + b + c
print(add(1, 2))
print(add(1, 2, 10))
# Expected output:
```

```
# 8
# 13
```

43. Extract int Type Values from a List of Heterogeneous Elements

```
items = [1, "2", 3.0, 4, True, "hello", 10]
ints_only = [x for x in items if type(x) is int]  # exclude bool (bool is a subclass of int)
print(ints_only)
# Expected output:
# [1, 4, 10]
```

44. Does Python Support Multiple Inheritance?

```
class Flyer:
    def fly(self):
        return "flying"
class Swimmer:
    def swim(self):
        return "swimming"
class Duck(Flyer, Swimmer):
    pass
d = Duck()
print(d.fly())
print(d.swim())
# Expected output:
# flying
# swimming
```

45. What is Monkey Patching?

```
class Greeter:
    def hello(self):
        return "hello"
g = Greeter()
```

```
print(g.hello())
def new_hello(self):
    return "hello (patched)"
# monkey patch the method
Greeter.hello = new_hello
print(g.hello())
print(Greeter().hello())
# Expected output:
# hello
# hello (patched)
# hello (patched)
```

46. Accept a Number and Check Whether It Is Prime or Not

```
def is_prime(n: int) -> bool:
def is_prime(n: int) -> bool:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    d = 3
    d = 3
        if n % d == 0:
            return False
        d += 2
    return True
for n in [1, 2, 3, 4, 17, 18, 19]:
    print(n, "prime" if is_prime(n) else "not prime")
# Expected output:
# 1 not prime
# 2 prime
# 3 prime
```

```
# 4 not prime
# 17 prime
# 18 not prime
# 19 prime
```

47. Print Whether a Number Is Odd or Even

```
def odd_or_even(n: int) -> str:
    return "even" if n % 2 == 0 else "odd"
for n in [0, 1, 2, 7, 10]:
    print(n, odd_or_even(n))
# Expected output:
# 0 even
# 1 odd
# 2 even
# 7 odd
# 10 even
```

48. Find Whether a Number Is Positive, Negative, or Zero

```
def sign(n: float) -> str:
    if n > 0:
        return "positive"
        return "positive"
        return "negative"
    return "zero"
for n in [3, -2.5, 0]:
    print(n, sign(n))
# Expected output:
# 3 positive
# -2.5 negative
# 0 zero
```

49. Find the Sum of Two Numbers

```
def add(a: int, b: int) -> int:
    return a + b
print(add(10, 25))
# Expected output:
# 35
```

50. Find GCD of Two Numbers

```
import math
a, b = 60, 48
print(math.gcd(a, b))
# Expected output:
# 12
```

51. Print the Following Pattern (Star Right Triangle)

```
n = 5
for i in range(1, n + 1):
    print("* " * i)
# Expected output:
# *
# * *
# * * *
# * * * *
# * * * * *
```

52. Print the Following Pattern (Number Triangle)

```
n = 5
for i in range(1, n + 1):
    for j in range(1, i + 1):
        print(j, end=" ")
    print()
# Expected output:
# 1
```

```
# 1 2
# 1 2 3
# 1 2 3 4
# 1 2 3 4 5
```

53. Print the Following Pattern (Sequential Numbers Triangle)

```
n = 5
num = 1
for i in range(1, n + 1):
    for _ in range(i):
        print(num, end=" ")
        num += 1
    print()
# Expected output:
# 1
# 2 3
# 4 5 6
# 7 8 9 10
# 11 12 13 14 15
```

54. Print the Following Pattern (Repeated Letters Triangle)

```
import string
n = 5
letters = string.ascii_uppercase
for i in range(1, n + 1):
    ch = letters[i - 1]
    print((ch + " ") * i)
# Expected output:
# A
# B B
# C C C
```

```
# D D D D
# E E E E E
```

55. Print the Following Pattern (Sequential Letters Triangle)

```
import string
n = 5
letters = iter(string.ascii_uppercase)
for i in range(1, n + 1):
    row = []
    for _ in range(i):
        row.append(next(letters))
    print(" ".join(row))
# Expected output:
# A
# B C
# D E F
# G H I J
# K L M N O
```

56. Print the Following Pattern (Alphabet Pyramid A..)

```
import string
n = 5
letters = string.ascii_uppercase
for i in range(1, n + 1):
    spaces = " " * (n - i)
    row = " ".join(letters[:i])
    print(spaces + row)
# Expected output:
#     A
#    A B
#   A B C
```

```
#  A B C D
# A B C D E
```

57. Print the Following Pattern (Centered Sequential Letters)

```
import string
n = 5
letters = iter(string.ascii_uppercase)
for i in range(1, n + 1):
    spaces = " " * (n - i)
    row = " ".join(next(letters) for _ in range(i))
    print(spaces + row)
# Expected output:
#     A
#    B C
#   D E F
#  G H I J
# K L M N O
```

58. Create a Pyramid with 5 Rows

```
n = 5
for i in range(n):
    spaces = " " * (n - i - 1)
    stars = "* " * (2 * i + 1)
    print(spaces + stars.rstrip())
# Expected output:
#     *
#    * * *
#   * * * * *
#  * * * * * * *
# * * * * * * * * *
```

59. Create a Sandglass (Hourglass) Pattern

```
n = 5
# top half (decreasing)
for i in range(n):
    spaces = " " * i
    stars = "* " * (n - i)
    print(spaces + stars.rstrip())
# bottom half (increasing), skip the middle line to
avoid duplicate
for i in range(n - 2, -1, -1):
    spaces = " " * i
    stars = "* " * (n - i)
    print(spaces + stars.rstrip())
# Expected output:
# * * * * *
#  * * * *
#   * * *
#    * *
#     *
#    * *
#   * * *
#  * * * *
# * * * * *
```

60. Create a Reverse Pyramid

```
n = 5
for i in range(n):
    spaces = " " * i
    stars = "* " * (n - i)
    print(spaces + stars.rstrip())
# Expected output:
# * * * * *
#  * * * *
#   * * *
```

```
#    * *
#     *
```

61. Create a Hill Triangle Pattern (Numbers)

This matches the image: 1, then 1 2 3, then 1..5, etc. Spacing uses two spaces to keep alignment with multi-digit numbers.

```
n = 5
for i in range(1, n + 1):
    spaces = " " * (n - i)
    row = " ".join(str(x) for x in range(1, 2 * i))
    print(spaces + row)
# Expected output:
#     1
#    1 2 3
#   1 2 3 4 5
#  1 2 3 4 5 6 7
# 1 2 3 4 5 6 7 8 9
```

62. Create the Following Pattern (Right-Aligned Descending Numbers)

```
n = 5
for i in range(n):
    spaces = "  " * i
    row = list(range(n - i, 0, -1))
    print(spaces + " ".join(map(str, row)))
# Expected output:
# 5 4 3 2 1
#   4 3 2 1
#     3 2 1
#       2 1
#         1
```

63. Create the Diamond Pattern (Numbers)

```
n = 5
# upper half (1..n)
for i in range(1, n + 1):
    spaces = " " * (n - i)
    row = " ".join([str(i)] * (2 * i - 1))
    print(spaces + row)
# lower half (n-1..1)
for i in range(n - 1, 0, -1):
    spaces = " " * (n - i)
    row = " ".join([str(i)] * (2 * i - 1))
    print(spaces + row)
# Expected output:
#     1
#    2 2 2
#   3 3 3 3 3
#  4 4 4 4 4 4 4
# 5 5 5 5 5 5 5 5 5
#  4 4 4 4 4 4 4
#   3 3 3 3 3
#    2 2 2
#     1
```

64. Print the Following Pattern (A, CC, EEE, ...)

```
import string
n = 5
letters = string.ascii_uppercase
# use odd-index letters: A(0), C(2), E(4), ...
for i in range(1, n + 1):
    ch = letters[2 * (i - 1)]
```

```
    print(" ".join([ch] * i))
# Expected output:
# A
# C C
# E E E
# G G G G
# I I I I I
```

Created by Igor Lukyanov, 02/16/2026 with mshell 1.4.1 (shell for AI and Mathematics)