Pocket Reference

Visit realpython.com to turbocharge your Python learning with in-depth Tutorials, real-world examples, and expert guidance.

🚀 Getting Started

Interacting With Python

```
$ python
>>> print("Hello, Python!")
Hello, Python!
```

Running a script

```
$ python my_script.py
```

Related Tutorials

- Python Basics: Start With the Python Interpreter
- Beginner's Guide: Python Basics Tutorials
- Run Python Scripts From the Command Line
- Command-Line Tutorials at Real Python

📦 Variables & Data Types

Basic Types

```
>>> 42            # int — Whole numbers
42
>>> 3.14          # float — Decimal numbers
3.14
>>> "Hello"       # str — Text/strings
'Hello'
>>> True          # bool — Boolean values
True
>>> None          # NoneType — Absence of value
```

Variable Assignment

```
name = "Leo"          # String
age = 7               # Integer
height = 5.6          # Float
is_cat = True         # Boolean
flaws = None          # None type
```

String Formatting

```
f_string = f"{name} is {age} years"
print(f_string)
# "Leo is 7 years"
```

Check Type and Conversion

```
type(age)             # <class 'int'>
isinstance(age, int)  # True

int("42")             # 42
float("3.14")         # 3.14
str(42)               # "42"
```

Related Tutorials

- Variables in Python: Usage and Best Practices
- Python Basics Tutorials (Types & Values)
- Python's Assignment Operator: Write Robust Assignments
- The Walrus Operator: Python's Assignment Expressions
- Python f-Strings: Effortless String Formatting
- Python String Formatting Best Practices
- Type Checking in Python

📝 Strings

String Basics

```
# Creating strings
single = 'Hello'
double = "World"
multi = """Multiple
line string"""

# String operations
greeting = "me" + "ow!"
repeat = "Meow!" * 3
length = len("Python")
```

String Methods

```
>>> "a".upper()                    # str.upper()
'A'
>>> "A".lower()                    # str.lower()
'a'
>>> " a ".strip()                  # str.strip()
'a'
>>> "abc".replace("bc", "ha")      # str.replace()
'aha'
>>> "a b".split()                  # str.split()
['a', 'b']
>>> "-".join(['a', 'b'])           # str.join()
'a-b'
```

String Indexing & Slicing

```
text = "Python"
text[0]      # 'P' (first)
text[-1]     # 'n' (last)
text[1:4]    # 'yth' (slice)
text[:3]     # 'Pyt' (from start)
text[3:]     # 'hon' (to end)
text[::2]    # 'Pto' (every 2nd)
text[::-1]   # 'nohtyP' (reverse)
```

Related Tutorials

- Strings and Character Data in Python
- String Tutorials on Real Python
- Essential String Methods in Python
- Python Slice Notation Explained

🔢 Numbers & Math

Arithmetic Operators

```
>>> 10 + 3     # Addition
13
>>> 10 - 3     # Subtraction
7
>>> 10 * 3     # Multiplication
30
>>> 10 / 3     # Division (float)
3.3333333333333335
>>> 10 // 3    # Floor division
3
>>> 10 % 3     # Modulo
1
>>> 2 ** 3     # Exponentiation
8
```

Useful Functions

```
abs(-5)              # 5
round(3.7)           # 4
round(3.14159, 2)    # 3.14
min(1, 2, 3)         # 1
max(1, 2, 3)         # 3
sum([1, 2, 3])       # 6
```

Type conversion

```
int("42")            # 42
float("3.14")        # 3.14
str(42)              # "42"
```

Related Tutorials

- Arithmetic Operators in Python
- Math and Numerical Computing Tutorials
- Python's Built-in Functions
- Converting Types in Python

📊 Collections

Creating Lists

```
nums = [5]
mixed = [1, "two", 3.0, True]
empty = []
```

List Methods

```
>>> nums = [5]                          # start list
>>> nums.append("x")                    # add to end
>>> nums
[5, 'x']
>>> nums.insert(0, "y")                 # insert at index 0
>>> nums
['y', 5, 'x']
>>> nums.extend(["z", 5])               # extend with iterable
>>> nums
['y', 5, 'x', 'z', 5]
>>> nums.remove("x")                    # remove first "x"
>>> nums
['y', 5, 'z', 5]
>>> last = nums.pop()                   # pop returns last element
>>> nums
['y', 5, 'z']
>>> last                                # popped value
5
```

List Indexing & Checks

```
fruits = ["banana", "apple", "orange"]
fruits[0]              # "banana"
fruits[-1]             # "orange"
"apple" in fruits      # True
len(fruits)            # 3
```

Tuples

```
# Creating Tuples
>>> point = (3, 4)
>>> single = (1,)          # Note the comma!
>>> empty = ()

# Tuple Unpacking
>>> point = (3, 4)
>>> x, y = point                # basic unpacking
>>> x
3
>>> y
4
>>> first, *rest = (1, 2, 3, 4) # extended unpacking
>>> first
1
>>> rest
[2, 3, 4]
```

Sets

```
# Creating Sets
>>> a = {1, 2, 3}
>>> b = set([3, 4, 4, 5])

# Set Operations
>>> a = {1, 2, 3}
>>> b = {3, 4, 5}
>>> a | b        # Union
{1, 2, 3, 4, 5}
>>> a & b        # Intersection
{3}
>>> a - b        # Difference
{1, 2}
>>> a ^ b        # Symmetric difference
{1, 2, 4, 5}
```

Dictionaries

```
# Creating Dictionaries
>>> pet = {"name": "Leo", "age": 42}
>>> empty = {}

# Dictionary Operations
>>> pet["sound"] = "Purr!"   # Add key and value
>>> pet["age"] = 7           # Update value
>>> age = pet.get("age", 0)  # Get with default
>>> del pet["email"]         # Delete key
>>> pet.pop("age")           # Remove & return

# Dictionary Methods
>>> pet = {"name": "Leo", "age": 7, "sound": "Purr!"}
>>> pet.keys()              # view keys
dict_keys(['name', 'age', 'sound'])
>>> pet.values()            # view values
dict_values(['Leo', 7, 'Purr!'])
>>> pet.items()             # key/value pairs
dict_items([('name', 'Leo'), ('age', 7), ('sound', 'Purr!')])
```

Related Tutorials

- Lists and Tuples in Python
- Common List Methods in Python
- Python Lists: Beyond the Basics
- Indexing and Slicing Lists in Python
- Membership Tests With `in` and `not in`
- Python Tuples: When to Use Them
- Unpacking in Python: Beyond the Basics
- Sets in Python
- Set Operations and Use Cases
- Dictionaries in Python
- Mastering Dictionaries in Python
- Dictionary Views: `keys()`, `values()`, and `items()`
- Data Structures Tutorials

🔀 Control Flow

If–Elif–Else

```
score = 85

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
else:
    grade = "F"
```

Comparison & Boolean

```
>>> 3 == 3              # Equal
True
>>> 3 != 4              # Not equal
True
>>> 2 < 5               # Less than
True
>>> 2 <= 2              # Less or equal
True
>>> (3 > 1) and (2 > 1)   # both True
True
>>> (3 > 5) or (2 < 4)    # either True
True
>>> not (3 == 4)        # opposite
True
```

Related Tutorials

- Conditional Statements in Python
- Comparison, Logical, and Identity Operators
- Truthy and Falsy in Python
- Python Basics Tutorials

🔁 Loops

Looping

```
# Loop 0--4
for i in range(5):
    print(i)

# Loop 1-5
for i in range(1, 5+1):
    print(i)

# Loop Over Collection
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

```
# Enumerate With Index
for i, fruit in enumerate(fruits):
    print(f"{i}: {fruit}")

# Zip Multiple Lists
names = ["Lyra", "Adrian"]
ages = [5, 3]
for name, age in zip(names, ages):
    print(f"{name} is {age}")
```

Break & Continue in while

```
while True:
    user_input = input("Enter 'quit' to exit: ")
    if user_input == "quit":
        break
    if user_input == "skip":
        continue
    print(user_input)
```

Related Tutorials

- Python `for` Loops (Definite Iteration)
- The `range()` Function
- Effective Use of `range()`
- Looping Patterns in Python
- Iterating in Python
- `enumerate()` in Python: Loop With Counters
- `zip()` in Python: Parallel Iteration
- Python `while` Loops (Indefinite Iteration)
- Control Flow: `break`, `continue`, and `pass`

🛠️ List Comprehensions

List Comprehensions

```
# Basic
squares = [x**2 for x in range(10)]

# With Condition
evens = [x for x in range(20) if x % 2 == 0]

# Nested
matrix = [[i*j for j in range(3)] for i in range(3)]
```

Other Comprehensions

```
# Dictionary Comprehension
word_lengths = {word: len(word) for word in ["hello", "world"]}

# Set Comprehension
unique_lengths = {len(word) for word in ["hi", "hello", "hey"]}
```

Related Tutorials

- List Comprehensions in Python
- Python `for` Loops vs Comprehensions
- Filter With Comprehensions
- Idiomatic Python: Comprehensions
- Nested Comprehensions
- Dictionary Comprehensions in Python
- Set Comprehensions in Python
- Data Structures Tutorials

🔧 Functions

No Parameters	`def greet(): ...`
One Parameter	`def greet(name): ...`
Default Parameter	`def greet(name="Everyone"): ...`

Multiple Return Values

```
def divide(a, b):
    quotient = a // b
    remainder = a % b
    return quotient, remainder

q, r = divide(10, 3)
```

Lambda Functions

```
square = lambda x: x**2
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
evens = list(filter(lambda x: x % 2 == 0, numbers))
```

Related Tutorials

- Defining Your Own Python Functions
- Functions Tutorial Index
- Returning Multiple Values in Python
- Tuple Unpacking and Assignment
- How to Use Python Lambda Functions
- Functional Tools: `map()` and `filter()`

File I/O

File Operations

```
# Read Entire File
with open("file.txt", mode="r") as file:
    content = file.read()

# Read File Line by Line
with open("file.txt", mode="r") as file:
    for line in file:
        print(line.strip())

# Write a File
with open("output.txt", mode="w") as file:
    file.write("Hello, World!\n")

# Append to a File
with open("log.txt", mode="a") as file:
    file.write("New log entry\n")
```

Related Tutorials

- Reading and Writing Files in Python
- The `open()` Function Explained
- Read Text Files Line by Line in Python
- Write to Files in Python
- Append to Files With Python
- File Modes and Context Managers

Error Handling

Try/Except/Else/Finally

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
except Exception as e:
    print(f"Error: {e}")
else:
    print("No errors occurred")
finally:
    print("This always runs")
```

Common exceptions

```
# ValueError
# IndexError
# KeyError
# FileNotFoundError
```

Related Tutorials

- Python Exceptions: An Introduction
- The `try`/`except`/`else`/`finally` Blocks
- Common Python Exceptions and How to Handle Them
- Exception Hierarchy and Best Practices

Working With JSON (and Dates)

Working With JSON

```
# JSON String to Dict
import json

data = json.loads('{"name": "Frieda", "age": 8}')

# Dict to JSON String
data_json = json.dumps({"name": "Frieda", "age": 8})

# Write Dict to JSON File
data = {"name": "Frieda", "age": 8}
with open("output.json", mode="w") as f:
    json.dump(data, f, indent=2)

# Load JSON File as Dict
with open("output.json", mode="r") as f:
    data = json.load(f)
```

Dates and Times

```
from datetime import datetime, date, timedelta

now = datetime.now()
today = date.today()
tomorrow = today + timedelta(days=1)

print(now.strftime("%Y-%m-%d %H:%M:%S"))  # 2025-07-31 20:01:16
```

Related Tutorials

- Working With JSON Data in Python
- Serialize and Deserialize JSON
- Write JSON to Files in Python
- Read JSON From Files in Python
- Python Datetime: Working With Dates and Times
- Mastering `strftime` and `strptime`

Miscellaneous

Truthy and Falsy

```
# Falsy Values
False
None
0
0.0
""
[]
{}
()

# Truthy List Check
if my_list:
    print(my_list)
```

Pythonic Constructs

```
# Swap Variables
a, b = b, a

# Flatten List
flat = [item for sublist in nested for item in sublist]

# Remove Duplicates, Preserve Order
unique = list(dict.fromkeys(my_list))

# Count Occurrences
from collections import Counter

counts = Counter(my_list)
```

Related Tutorials

- Truth Value Testing in Python
- Conditional Statements and Truthiness
- Idiomatic Python: EAFP and Truthy Checks
- Multiple Assignment and Unpacking
- Flatten Lists in Python (Comprehensions & More)
- Remove Duplicates From Lists in Python
- Counter in Python: Tally Hashable Objects
- Pythonic Tips & Tricks

Virtual Environments

Create Virtual Environment

```
$ python -m venv .venv
```

Activate Virtual Environment

```
PS> .venv\Scripts\activate
```

Activate Virtual

```
$ source .venv/bin/activate
```

Deactivate Virtual

```
(.venv) $ deactivate
```

Related Tutorials

- Python Virtual Environments: A Primer
- Manage Multiple Python Environments
- Activate and Use Virtual Environments
- Virtual Environment Management Tips

Packages

Install Packages

```
$ python -m pip install requests
```

Save Requirements & Install

```
$ python -m pip freeze > requirements.txt
$ python -m pip install -r requirements.txt
```

Related Tutorials

- What Is `pip`?
- Installing Python Packages
- Requirements Files in Python Projects
- Pinning and Reproducible Environments
- Install From a Requirements File
- Packaging & Distribution Tutorials

Links

Visit Real Python

- Real Python Search: realpython.com/search
- The Real Python Podcast: realpython.com/podcasts/
- Common Python Terms: realpython.com/ref/
- Learning Paths: realpython.com/learning-paths/
- Quizzes and Exercises: realpython.com/quizzes/
- Code Mentor: realpython.com/mentor/

Find Real Python on Social Media

- X/Twitter: x.com/realpython
- Instagram: instagram.com/realpython
- YouTube: youtube.com/realpython
- Facebook: facebook.com/LearnRealPython
- LinkedIn: linkedin.com/company/realpython-com
- Mastodon: fosstodon.org/@realpython

Contact

info@realpython.com