Python Cheat Sheet

This cheat sheet provides a quick reference for essential Python concepts and syntax.

Basic Data Types

Data Type	Description	Example
`int`	Integer (whole number)	`x = 5`
`float`	Floating-point number (decimal)	`y = 3.14`
`str`	String (sequence of characters)	`z = "Hello"`
`bool`	Boolean (True or False)	`a = True`
`list`	Ordered, mutable collection	`b = [1, 2, "three"]`
`tuple`	Ordered, immutable collection	`c = (4, 5, 6)`
`dict`	Unordered, mutable key-value pairs	`d = {"name": "Alice", "age": 30}`
`set`	Unordered, mutable collection of unique elements	`e = {1, 2, 3}`
`None`	Represents the absence of a value	`x = None`

Note: Lists and dictionaries are mutable (can be changed); tuples and strings are immutable (cannot be changed).

Operators

Operator	Description	Example
`+`	Addition	`3 + 5  # Output: 8`
`-`	Subtraction	`10 - 4  # Output: 6`
`*`	Multiplication	`2 * 6  # Output: 12`

Operator	Description	Example
`/`	Division	`15 / 3  # Output: 5.0`
`//`	Floor division (division that rounds down)	`7 // 2  # Output: 3`
`%`	Modulus (remainder)	`10 % 3  # Output: 1`
`**`	Exponentiation	`2 ** 3  # Output: 8`
`==`	Equal to	`5 == 5  # Output: True`
`!=`	Not equal to	`5 != 3  # Output: True`
`<`	Less than	`4 < 7  # Output: True`
`>`	Greater than	`10 > 2  # Output: True`
`<=`	Less than or equal to	`5 <= 5  # Output: True`
`>=`	Greater than or equal to	`8 >= 6  # Output: True`
`and`	Logical AND	`x = 5; y = -3; x > 0 and y < 10  # Output: True`
`or`	Logical OR	`x = -1; y = 5; x > 0 or y > 0  # Output: True`
`not`	Logical NOT	`x = 5; not x > 0  # Output: False`

Control Flow

- **if/elif/else:**

```
if x > 0:
```

```
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

- **for loop:**

```
for i in range(5):
    print(i)  # Outputs: 0, 1, 2, 3, 4
```

- **while loop:**

```
while x < 10:
    print(x)
    x += 1
```

- **break:** Exits the loop entirely

```
for i in range(5):
    if i == 3:
        break
    print(i)  # Outputs: 0, 1, 2
```

- **continue:** Skips the rest of the current iteration

```
for i in range(5):
    if i == 3:
        continue
    print(i)  # Outputs: 0, 1, 2, 4
```

- **pass:** Does nothing (placeholder)

```
for i in range(5):
    pass  # Loop does nothing
```

Note: Python uses indentation (typically 4 spaces) to define code blocks.

Functions

- **Basic function:**

```
def greet(name):
    print("Hello, " + name)
greet("Alice")  # Outputs: Hello, Alice
```

- **Default argument:**

```
def greet(name="World"):
    print("Hello, " + name)
greet()  # Outputs: Hello, World
```

- **Return statement:**

```python
def add(a, b):

    return a + b

result = add(2, 3)  # result = 5
```

Built-in Functions

- `print()` - Prints to the console: `print("Hello")`
- `len()` - Returns the length of an object: `len([1, 2, 3])  # Output: 3`
- `type()` - Returns the type of an object: `type(5)  # Output: <class 'int'>`
- `input()` - Gets input from the user: `name = input("Enter name: ")`
- `range()` - Generates a sequence of numbers: `list(range(5))  # Output: [0, 1, 2, 3, 4]`
- `min()` - Returns the smallest value: `min(1, 2, 3)  # Output: 1`
- `max()` - Returns the largest value: `max(1, 2, 3)  # Output: 3`
- `sum()` - Sums an iterable: `sum([1, 2, 3])  # Output: 6`
- `int()` - Converts to integer: `int("5")  # Output: 5`
- `str()` - Converts to string: `str(123)  # Output: "123"`

String Methods

- `str.upper()` - Converts to uppercase: `"hello".upper()  # Output: "HELLO"`
- `str.lower()` - Converts to lowercase: `"HELLO".lower()  # Output: "hello"`
- `str.strip()` - Removes leading/trailing whitespace: `"  hi  ".strip()  # Output: "hi"`
- `str.split()` - Splits into a list: `"a,b,c".split(",")  # Output: ["a", "b", "c"]`
- `str.join()` - Joins a list into a string: `",".join(["a", "b"])  # Output: "a,b"`
- `str.replace()` - Replaces a substring: `"Hello".replace("H", "J")  # Output: "Jello"`
- **f-string:** Modern formatting: `name = "Alice"; f"Hi, {name}"  # Output: "Hi, Alice"`

Note: String slicing: `text[0:3]` extracts "Hel" from "Hello".

List Methods

- `list.append()` - Adds an element: `[1, 2].append(3)  # Output: [1, 2, 3]`
- `list.insert()` - Inserts at index: `[1, 3].insert(1, 2)  # Output: [1, 2, 3]`
- `list.remove()` - Removes first occurrence: `[1, 2, 2].remove(2)  # Output: [1, 2]`
- `list.pop()` - Removes and returns element: `[1, 2].pop(1)  # Output: 2; list = [1]`
- `list.sort()` - Sorts the list: `[3, 1, 2].sort()  # Output: [1, 2, 3]`
- `list.index()` - Returns index of value: `[1, 2, 3].index(2)  # Output: 1`
- **List comprehension:** `[x**2 for x in range(5)]  # Output: [0, 1, 4, 9, 16]`

Dictionary Methods

- `dict.keys()` - Returns keys: `d.keys()  # Output: dict_keys(['name', 'age'])`
- `dict.values()` - Returns values: `d.values()  # Output: dict_values(['Alice', 30])`
- `dict.items()` - Returns key-value pairs: `d.items()  # Output: dict_items([('name', 'Alice'), ('age', 30)])`
- `dict.get()` - Returns value for key:
- `d.get("name")  # Output: "Alice"`
  `dict.update()` - Updates with another dict: `d.update({"key": "value"})`

File I/O

- **Reading:**

```
with open("myfile.txt", "r") as f:

    content = f.read()  # Reads entire file

    lines = f.readlines()  # Reads lines into a list
```

- **Writing:**

```
with open("myfile.txt", "w") as f:

    f.write("New content")
```

- **Modes:** `"r"` (read), `"w"` (write), `"a"` (append), `"rb"` (read binary)

Modules

- **math:** Mathematical functions

```
import math

math.sqrt(16)  # Output: 4.0

math.ceil(3.2)  # Output: 4

math.pi  # Output: 3.14159...
```

- **random:** Random number generation

```
import random

random.randint(1, 10)  # e.g., Output: 7

random.choice([1, 2, 3])  # e.g., Output: 2
```

- **datetime:** Date and time manipulation

```
import datetime

datetime.datetime.now()  # e.g., Output: 2025-02-21 12:34:56

datetime.date.today()  # e.g., Output: 2025-02-21
```

Error Handling

- **try/except:**

```
try:

    x = 1 / 0

except ZeroDivisionError:

    print("Cannot divide by zero")
```

Comments

- Single-line: `# This is a comment`
- Multi-line: `"""This is a multi-line comment"""`

Glossary for Python Cheat Sheet

- **Append**: Add something to the end of a list.
- **Argument**: A value you give to a function to use, like telling it what to work with (e.g., `"Alice"` in `greet("Alice")`).
- **Boolean**: A type of data that can only be `True` or `False`, like a yes-or-no answer.
- **Collection**: A group of items stored together, like a list or set.
- **Comprehension**: A short way to create a list or dictionary by looping through items in one line of code.
- **Console**: The screen or area where a program shows its output or messages.
- **Convert**: Change one type of data into another, like turning a number into a string.
- **Decimal**: A number with a dot (like 3.14), also called a floating-point number.
- **Default**: A preset value used if you don't provide one, like a backup option.
- **Define**: Set up or create something in code, like making a function with `def`.
- **Dictionary**: A way to store data using pairs (keys and values), like a word and its meaning in a real dictionary.

- **Element**: One item in a collection, like a single number in a list.
- **Exponentiation**: Raising a number to a power, like 2^3 (2 ** 3 = 8).
- **Extract**: Pull out a specific part of something, like getting letters from a string.
- **File I/O**: Reading from or writing to files on your computer (I/O means "input/output").
- **Float**: A number with a decimal point, like 3.14.
- **Floor Division**: Division that rounds down to the nearest whole number, ignoring the decimal part.
- **Format**: Arrange or shape text in a specific way, like adding a name into a sentence.
- **Function**: A reusable block of code that does a specific task, like printing a greeting.
- **Generate**: Create or produce something, like making a list of numbers.
- **Immutable**: Cannot be changed after it's created (e.g., tuples and strings).
- **Import**: Bring in extra tools or features from a module to use in your code.
- **Indentation**: Spaces at the start of a line of code that Python uses to group instructions together.
- **Index**: The position of an item in a list or string, starting at 0 (e.g., first item is at index 0).
- **Insert**: Put something into a specific spot in a list.
- **Integer**: A whole number without a decimal, like 5 or -3.
- **Iterable**: Something you can loop through, like a list or range of numbers.
- **Iteration**: One round of a loop, like going through a list item by item.
- **Key**: The label in a dictionary that connects to a value, like "name" in `{"name": "Alice"}`.
- **Length**: How many items are in something, like the number of letters in a string.
- **Logical**: Related to reasoning or true/false conditions, like checking if something is bigger than another thing.
- **Loop**: A way to repeat code multiple times, like a `for` or `while` loop.
- **Method**: A special function that belongs to a type of data, like `upper()` for strings.
- **Module**: A file with extra tools or code you can use by importing it, like `math` or `random`.
- **Modulus**: The remainder after division, like 10 % 3 = 1 because 3 goes into 10 three times with 1 left over.
- **Mutable**: Can be changed after it's created (e.g., lists and dictionaries).

- **Object**: A piece of data in Python, like a number, string, or list.
- **Operator**: A symbol that does something, like `+` for adding or `==` for checking if things are equal.
- **Ordered**: Items stay in the same sequence, like in a list or tuple.
- **Output**: What the program shows or produces, like text on the screen.
- **Pairs**: Two things connected together, like a key and value in a dictionary.
- **Placeholder**: Something that holds a spot but doesn't do anything, like `pass`.
- **Random**: Something unpredictable, like picking a number by chance.
- **Range**: A sequence of numbers, like 0 through 4.
- **Remove**: Take something out of a list or collection.
- **Return**: Give a result back from a function to use later.
- **Sequence**: A set of items in a specific order, like a list or string.
- **Slicing**: Cutting out a piece of a string or list, like taking the first three letters.
- **Sort**: Arrange items in order, like smallest to biggest.
- **Statement**: A complete instruction in code, like `print("Hello")`.
- **String**: A group of characters, like letters or words (e.g., "Hello").
- **Substring**: A smaller part of a string, like "Hel" from "Hello".
- **Syntax**: The rules for writing code correctly so Python understands it.
- **Tuple**: A collection of items that can't be changed, like a sealed list.
- **Unique**: No duplicates allowed, like in a set.
- **Unordered**: Items don't have a fixed sequence, like in a set or dictionary.
- **Update**: Change or add to something, like adding new pairs to a dictionary.
- **Value**: The data tied to a key in a dictionary, like "Alice" for "name".
- **Whitespace**: Empty spaces or tabs in text or code, like spaces before or after a word.