The Ultimate

# PYTHON FUNCTIONS CHEATSHEET

Save this before your next AI project.

## Basic Functions

### 1 print()

Outputs text, variables, or any data to the console. Accepts multiple arguments separated by commas and automatically adds spaces between them.

EXAMPLE

```
print("Welcome to Python!")
print("Name:", name, "Age:", age)
print(f"Total: ${price:.2f}")
```

### 2 input()

Pauses program execution and waits for user to type something. Always returns a string, even if user enters numbers.

EXAMPLE

```
name = input("Enter your name: ")
age = int(input("Enter age: "))
```

# Basic Functions

## 3 len()

Returns the number of items in a sequence (string, list, tuple) or collection (set, dict). For dictionaries, it counts key-value pairs.

EXAMPLE

```
len("Python") # 6 characters
len([1, 2, 3]) # 3 elements
len({"a": 1, "b": 2}) # 2 key-value pairs
```

## 4 type()

Returns the class/data type of any object. Helpful for debugging and understanding what kind of data you're working with.

EXAMPLE

```
type(10) #<class 'int'>
type("Hello") # <class 'str'>
type([1, 2]) # <class 'list'>
```

# Basic Functions

## 5 int()

Converts strings or floats to integers. Truncates decimals (doesn't round). Can also convert from different number bases.

EXAMPLE

```
int("100") #String to int: 100
int(3.9) #Float to int: 3
int("1010", 2) # Binary string to int: 10
int("FF", 16) # Hex string to int: 255
```

## 6 float()

Converts strings or integers to floating-point (decimal) numbers. Essential for precise calculations involving decimals.

EXAMPLE

```
float("3.14") # String to float: 3.14
float(5) #Int to float: 5.0
float("1.5e2") # Scientific notation: 150.0
```

# Basic Functions

## 7 str()

Converts any data type to its string representation. Allows you to concatenate non-string values with strings.

EXAMPLE

```
str(100) #Int to string: "100"
"Age: " + str(25) # Concatenation: "Age: 25"
str([1, 2]) #List to string: "[1, 2]"
```

## 8 bool()

Converts any value to Boolean (True/False). Returns False for empty/zero values, True for everything else.

EXAMPLE

```
bool(1) # True (non-zero)
bool(0) # False (zero)
bool("") # False (empty string)
bool("Hi") # True (non-empty)
```

# Basic Functions

## 9 list()

Converts iterables (strings, tuples, sets) into a list. Creates a mutable sequence that can be modified after creation.

EXAMPLE

```
list("abc")  #String to list: ['a', 'b', 'c']
list((1, 2, 3)) # Tuple to list: [1, 2, 3]
list(range(5)) # Range to list: [0, 1, 2, 3, 4]
```

---

## 10 tuple()

Converts iterables to an immutable tuple. Once created, elements cannot be added, removed, or changed. Uses less memory than lists.

EXAMPLE

```
tuple([1, 2, 3]) # List to tuple: (1, 2, 3)
tuple("abc")  #String to tuple: ('a', 'b', 'c')
x, y = tuple([10, 20) # Unpacking
```

# Basic Functions

## 11 set()

Creates an unordered collection of unique elements. Automatically removes duplicates and provides fast membership testing.

EXAMPLE

```
set([1, 1, 2, 2, 3]) # Duplicates removed: {1, 2, 3}
set("hello") #Unique chars: {'h', 'e', 'l', 'o'}
{1, 2} & {2, 3} #Set intersection: {2}
```

## 12 dict()

Creates a dictionary (key-value mapping). Provides fast lookups by key. Keys must be immutable (strings, numbers, tuples).

EXAMPLE

```
dict(name="John", age=30) # {'name': 'John', 'age': 30}
dict([("a", 1), ("b", 2)]) # From list of tuples
dict(zip(keys, values)) # Combine two lists
```

# Basic Functions

## 13 abs()

Returns the absolute value (distance from zero). Removes negative sign, always returns positive number or zero.

EXAMPLE

```
abs(-5) # 5
abs(5) # 5
abs(-3.14) # 3.14
abs(10 - 20) # 10 (difference)
```

---

## 14 round()

Rounds numbers to specified decimal places. Default is 0 decimals (integer). Negative values round to left of decimal.

EXAMPLE

```
round(3.6) #4(nearest integer)
round(3.14159, 2) # 3.14 (2 decimals)
round(1250, -2) # 1200 (nearest hundred)
```

# Basic Functions

## 15 max()

Returns the largest item from an iterable or the largest of multiple arguments. Works with numbers, strings (alphabetically).

EXAMPLE

```
max([1, 4, 2]) # 4
max(5, 10, 3, 8) #10
max("apple", "zebra") # "zebra"
```

---

## 16 min()

Returns the smallest item from an iterable or the smallest of multiple arguments. Opposite of max().

EXAMPLE

```
min(3, 1, 2) # 1
min([5, 2, 8]) # 2
min("apple", "ant") # "ant"
```

# Basic Functions

## 17 sum()

Adds all numeric elements in an iterable. Can optionally add a start value. Only works with numbers, not strings.

EXAMPLE

```python
sum([1, 2, 3]) # 6
sum([1, 2, 3], 10) # 16 (starts from 10)
sum(range(1, 11)) # 55 (sum of 1-10)
```

## 18 sorted()

Returns a NEW sorted list from any iterable. Original remains unchanged. Can sort in reverse or with custom key function.

EXAMPLE

```python
sorted([3, 1, 2]) # [1, 2, 3]
sorted("python") #['h','n','o', 'p', 't', 'y']
sorted([3, 1, 2], reverse=True) # [3, 2, 1]
```

# Basic Functions

## 19 range()

Generates a sequence of numbers. Memory efficient - doesn't create full list. Takes start, stop, step parameters.

EXAMPLE

```
range(5) # 0, 1, 2, 3, 4
range(2, 8) # 2, 3, 4, 5, 6, 7
range(0, 10, 2) # 0, 2, 4, 6, 8
range(10, 0, -1) # 10, 9, 8...1
```

## 20 enumerate()

Adds counter/index to an iterable. Returns pairs of (index, value). Start parameter sets initial counter value.

EXAMPLE

```
for i, name in enumerate(["Alice", "Bob"]):
    print(f"{i}: {name}") # 0: Alice, 1: Bob

list(enumerate(["a", "b"], start=1)) # [(1,'a'), (2,'b')]
```

# Intermediate Functions

## 21 map()

Applies a function to every item in an iterable. Returns a map object (iterator). Often converted to list for viewing.

EXAMPLE

```
list(map(str, [1, 2, 3])) #['1','2', '3']
list(map(len, ["a", "bb"])) # [1, 2]
list(map(lambda x: x**2, [1, 2, 3])) # [1, 4, 9]
```

## 22 filter()

Filters items based on a condition function. Keeps only items where function returns True. Returns iterator.

EXAMPLE

```
list(filter(lambda x: x > 2, [1, 2, 3, 4])) # [3, 4]
list(filter(lambda x: x % 2 == 0, range(10))) # Even
numbers
list(filter(None, [0, 1, "", "hi"])) # Removes falsy values
```

# Intermediate Functions

## 23 zip()

Pairs up elements from multiple iterables. Stops when shortest iterable is exhausted. Returns iterator of tuples.

EXAMPLE

```
list(zip([1, 2], ["a", "b"])) # [(1, 'a'), (2, 'b')]
list(zip([1, 2, 3], ["a", "b"])) # [(1, 'a'), (2, 'b')]
dict(zip(["a", "b"], [1, 2])) # {'a': 1, 'b': 2}
```

---

## 24 sorted(key)

Sorts with custom comparison. Key function extracts comparison value from each element.

EXAMPLE

```
sorted(["a", "bb", "ccc"], key=len) # By length
sorted(["Apple", "banana"], key=str.lower) # Case-
insensitive
sorted([{"age": 30}, {"age": 20}], key=lambda x: x["age"])
```

# Intermediate Functions

## 25 ord()

Returns the Unicode code point (integer) of a single character. Useful for character comparisons and encoding.

EXAMPLE

```
ord("A") # 65 (ASCII/Unicode value)
ord("a") # 97
ord("€") # 8364 (Euro symbol)
```

## 26 chr()

Converts Unicode code point (integer) back to its character. Inverse operation of ord().

EXAMPLE

```
chr(65) #  'A'
chr(97) # 'a'
chr(8364) # '€'
```

# Intermediate Functions

## 27 eval()

Evaluates a string as a Python expression and returns the result. DANGEROUS with untrusted input!

EXAMPLE

```
eval("2 + 2") # 4
eval("len('hello')") # 5
x = 10
eval("x * 2") # 20
```

## 28 hex()

Converts an integer to a lowercase hexadecimal string prefixed with "0x".

EXAMPLE

```
hex(255) # '0xff'
hex(16) # '0x10'
hex(0) # '0x0'
```

# Intermediate Functions

## 29 bin()

Converts an integer to a binary string prefixed with "0b". Base-2 representation.

EXAMPLE

```
bin(10) # '0b1010'
bin(255) # '0b11111111'
bin(0) # '0b0'
```

## 30 oct()

Converts an integer to an octal string prefixed with "0o". Base-8 representation.

EXAMPLE

```
oct(8) # '0o10'
oct(64) # '0o100'
oct(0) # '0o0'
```

# Intermediate Functions

## 31 isinstance()

Checks if an object is an instance of a class or tuple of classes. Returns True/False. Safer than type().

EXAMPLE

```
isinstance(5, int) # True
isinstance("hi", str) # True
isinstance(5, (int, float)) # True
```

## 32 issubclass()

Checks if a class is a subclass of another class. Returns True/False. Both arguments must be classes.

EXAMPLE

```
issubclass(bool, int) # True
issubclass(int, object) # True
issubclass(str, int) # False
```

# Intermediate Functions

## 33 id()

Returns unique integer identifier for an object's memory address. Two objects with same id are the exact same object.

EXAMPLE

```python
id(5) # 140712345678912
x = [1, 2]
y = x
id(x) == id(y) # True (same object)
```

## 34 hash()

Returns hash value (integer) of an object. Used for fast dictionary lookups. Only works with immutable objects.

EXAMPLE

```python
hash("hello") # 5994345897359492
hash(42) #42
hash((1, 2)) # 3713081631934410656
```

# Intermediate Functions

## 35 dir()

Returns sorted list of all attributes and methods of an object. Without argument, lists names in current scope.

EXAMPLE

```
dir([]) # ['append', 'clear', 'copy', ...]
dir(str) # ['capitalize', 'center', ...]
dir() # Lists all names in current scope
```

---

## 36 getattr()

Gets attribute value from an object by name (string). Can provide default if attribute doesn't exist.

EXAMPLE

```
getattr(str, "upper") # <method 'upper'>
getattr([1, 2], "append") #<method 'append'>
getattr(obj, "name", "Unknown") # Default value
```

# Intermediate Functions

## 37 setattr()

Sets an attribute on an object using a string name. Dynamically adds or modifies attributes.

EXAMPLE

```python
class Obj: pass
obj = Obj()
setattr(obj, "name", "John") # obj.name = "John"
setattr(obj, "age", 30) #obj.age = 30
```

---

## 38 hasattr()

Checks if an object has a specific attribute (by name). Returns True/False. Safer than try/except.

EXAMPLE

```python
hasattr([], "append") # True
hasattr([], "xyz") # False
hasattr("hi", "upper") # True
```

# Intermediate Functions

## 39 globals()

Returns dictionary of all global variables and their values in current module. Modifications affect global scope.

EXAMPLE

```python
x = 10
globals()["x"] #10
globals()["x"] = 20 # Changes x to 20
"x" in globals() # True
```

## 40 locals()

Returns dictionary of all local variables in current scope. Snapshot only - modifications don't affect actual variables.

EXAMPLE

```python
def func():
    x = 10
    print(locals()) # {'x': 10}
```

# Intermediate Functions

## 41 open()

Opens a file and returns a file object. Modes: "r" (read), "w" (write), "a" (append), "b" (binary). Use with context manager.

EXAMPLE

```
with open("data.txt", "r") as f:
    content = f.read()

with open("data.txt", "w") as f:
    f.write("Hello")
```

## 42 repr()

Returns "official" string representation of object. Designed for developers. Shows how to recreate the object.

EXAMPLE

```
repr([1, 2, 3]) #'[1, 2, 3]'
repr("hello") #"'hello'" (includes quotes)
repr(datetime.now()) # 'datetime.datetime(2024,1,1,  ...)'
```

# Intermediate Functions

## 43 format()

Formats a value using a format specifier. Less common than f-strings but useful for dynamic formatting.

EXAMPLE

```
format(10.5, ".1f") # '10.5' (1 decimal)
format(42,   "05d")# '00042' (pad with zeros)
format(1234, ",") # '1,234' (thousands separator)
format(0.5, ".2%") # '50.00%' (percentage)
```

## 44 slice()

Creates a slice object for advanced indexing. Alternative to [start:stop:step] syntax. Reusable.

EXAMPLE

```
s = slice(1, 5, 2) 5]
[0, 1, 2, 3, 4, [s]   # [1, 3]
"Python"[slice(0,3)]   # 'Pyt'
s = slice(None, -1) # Everything except last
```

# Intermediate Functions

## 45 reversed()

Returns a reverse iterator. Doesn't modify original. Memory efficient - doesn't create new list unless converted.

EXAMPLE

```python
list(reversed([1, 2, 3])) #[3, 2, 1]
list(reversed("hello")) #['o', 'l', 'l', 'e', 'h']
for item in reversed(range(5)): # 4, 3, 2, 1, 0
    print(item)
```

## 46 super()

Returns proxy object allowing access to parent class methods. Used in inheritance to call overridden methods.

EXAMPLE

```python
class Parent:
    def greet(self): return "Hello"

class Child(Parent):
    def greet(self):
        return super().greet() + " World"
```

# Intermediate Functions

## 47 staticmethod()

Decorator that defines a method that doesn't receive instance (self) or class (cls) as first argument.

EXAMPLE

```
class Math:
    @staticmethod
    def add(x, y): # No self parameter
        return x + y

Math.add(5, 3) # Can call without instance
```

## 48 classmethod()

Decorator that defines a method receiving class (cls) as first argument instead of instance (self).

EXAMPLE

```
class    Person:
       count  =  0
    @classmethod
    def increment(cls):
```

# Advanced Functions

## 49 property()

Decorator that defines managed attributes with getter, setter, deleter. Allows method-like behavior with attribute syntax.

EXAMPLE

```
class Circle:
    def __init__(self, radius):
        self._radius = radius

    @property
    def diameter(self):
        return self._radius * 2

c = Circle(5)
c.diameter #10
```

## 50 compile()

Compiles Python source code (string) into code object. Can be executed with exec() or eval(). Modes: eval, exec, single.

EXAMPLE

# Advanced Functions

## 51 exec()

Executes dynamically created Python code (string or compiled code). Can execute statements, not just expressions.

EXAMPLE

```
exec("x = 10") # Creates variable x
exec("for i in range(3): print(i)") # Executes loop
code = "def func(): return 42"
exec(code) # Defines function
```

## 52 complex()

Creates complex number with real and imaginary parts. Used in scientific/engineering calculations.

EXAMPLE

```
complex(2, 3) # (2+3j)
complex(5) # (5+0j)
z = 3 + 4j
abs(z) # 5.0 (magnitude)
```

# Advanced Functions

## 53 frozenset()

Creates immutable version of set. Can be used as dictionary key or set element. Cannot add/remove items after creation.

EXAMPLE

```python
fs = frozenset([1, 2, 3])
{frozenset([1, 2]): "value"} # Can use as dict key
fs2 = frozenset([3, 4])
fs | fs2 #Union:frozenset({1, 2, 3, 4})
```

## 54 bytearray()

Creates mutable array of bytes. Like bytes but can be modified. Used for binary data manipulation.

EXAMPLE

```python
ba = bytearray(3) # bytearray(b'\x00\x00\x00')
ba[0] = 65 # bytearray(b'A\x00\x00')
ba.append(66) # bytearray(b'A\x00\x00B')
```

# Advanced Functions

## 55 bytes()

Creates immutable sequence of bytes. Used for binary data, file I/O, network protocols. Cannot be modified.

EXAMPLE

```
bytes(3) # b'\x00\x00\x00'
bytes([65, 66, 67]) # b'ABC'
bytes("hello", "utf-8") # b'hello'
b"hello"[0] #104(byte value)
```

## 56 memoryview()

Creates memory view object allowing access to internal buffers without copying. Efficient for large binary data.

EXAMPLE

```
data = bytearray(b"hello")
mv = memoryview(data)
mv[0] = 72 #Changesdata to b"Hello"
mv.tobytes() # b'Hello'
```

# Advanced Functions

## 57 callable()

Checks if object can be called like a function (has __call__ method). Returns True for functions, methods, classes.

EXAMPLE

```
callable(print) # True
callable(5) # False
callable(lambda x: x) # True
callable(str) # True
```

## 58 delattr()

Deletes an attribute from an object by name (string). Equivalent to del obj.attr. Raises AttributeError if doesn't exist.

EXAMPLE

```
class Obj:     x =
10   obj   =   Obj()
delattr(obj, "x")
                  # Deletes obj.x
```

# Advanced Functions

## 59 next()

Retrieves next item from an iterator. Can provide default value to return when iterator is exhausted.

EXAMPLE

```
it = iter([1, 2, 3])
next(it)     #     1
next(it) # 2
next(it) # 3
next(it, "done") # "done"
```

## 60 all()

Returns True if all elements in iterable are truthy (or iterable is empty). Short-circuits on first False.

EXAMPLE

```
all([True, True, True]) # True
all([True, False, True]) # False
all([]) # True (empty)
all([1, 2, 3]) # True
```

# Want more content like this?

Tap that follow button and stay in the loop!

Like Comment Share Save