Python Cheat Sheet: Keywords

"A puzzle a day to learn, code, and play" ➜ Visit finxter.com

Keyword	Description	Code example
False, True	Data values from the data type Boolean	`False == (1 > 2), True == (2 > 1)`

and, or, not	Logical operators:
	(x and y) ➜ both x and y must be True
	(x or y) ➜ either x or y must be True
	(not x) ➜ x must be false
		`x, y = True, False`
		`(x or y) == True        # True`
		`(x and y) == False      # True`
		`(not y) == True         # True`

break	Ends loop prematurely	`while(True):`
		`    break # no infinite loop`
		`print("hello world")`

continue	Finishes current loop iteration	`while(True):`
		`    continue`
		`    print("43") # dead code`

class
 	
def
	Defines a new class ➜ a real-world concept (object oriented programming)
	
	Defines a new function or class method. For latter, first parameter ("self") points to the class object. When calling class method, first parameter is implicit.
		`class Beer:`
		`    def __init__(self):`
		`        self.content = 1.0`
		`    def drink(self):`
		`        self.content = 0.0`
		
		`becks = Beer() # constructor - create class`
		`becks.drink() # beer empty: b.content == 0`

if, elif, else	Conditional program execution: program starts with "if" branch, tries the "elif" branches, and finishes with "else" branch (until one branch evaluates to True).	`x = int(input("your value: "))`
		`if x > 3: print("Big")`
		`elif x == 3: print("Medium")`
		`else: print("Small")`

for, while	`# For loop declaration`
	`for i in [0,1,2]:`
	`    print(i)`
		`# While loop - same semantics`
		`j = 0`
		`while j < 3:`
		`    print(j)`
		`    j = j + 1`

in	Checks whether element is in sequence	`42 in [2, 39, 42] # True`

is	Checks whether both elements point to the same object	`y = x = 3`
		`x is y # True`
		`[3] is [3] # False`

None	Empty value constant	`def f():`
		`    x = 2`
		`f() is None # True`

lambda	Function with no name (anonymous function)	`(lambda x: x + 3)(3) # returns 6`

return	Terminates execution of the function and passes the flow of execution to the caller. An optional value after the return keyword specifies the function result.	`def incrementor(x):`
		`    return x + 1`
		`incrementor(4) # returns 5`

finxter

Python Cheat Sheet: Basic Data Types

"A puzzle a day to learn, code, and play" ➜ Visit finxter.com

	Description	Example
| **Boolean** | The Boolean data type is a truth value, either True or False.<br><br>The Boolean operators ordered by priority:<br>not x ➜ "if x is False, then x, else y"<br>x and y ➜ "if x is False, then x, else y"<br>x or y ➜ "if x is False, then y, else x"<br><br>These comparison operators evaluate to True:<br>1 < 2 and 0 <= 1 and 3 > 2 and 2 >=2 and 1 == 1 and 1 != 0 # True | ```python
## 1. Boolean Operations
x, y = True, False
print(x and not y) # True
print(not x and y or x) # True

## 2. If condition evaluates to False
if None or 0 or 0.0 or '' or [] or {} or set():
    # None, 0, 0.0, empty strings, or empty
    # container types are evaluated to False
    print("Dead code") # Not reached
``` |
| **Integer, Float** | An integer is a positive or negative number without floating point (e.g. 3). A float is a positive or negative number with floating point precision (e.g. 3.14159265359).<br><br>The '//' operator performs integer division. The result is an integer value that is rounded toward the smaller integer number (e.g. 3 // 2 == 1). | ```python
## 3. Arithmetic Operations
x, y = 3, 2
print(x + y) # = 5
print(x - y) # = 1
print(x * y) # = 6
print(x / y) # = 1.5
print(x // y) # = 1
print(x % y) # = 1s
print(-x) # = -3
print(abs(-x)) # = 3
print(int(3.9)) # = 3
print(float(3)) # = 3.0
print(x ** y) # = 9
``` |
| **String** | Python Strings are sequences of characters.<br><br>The four main ways to create strings are the following.<br><br>1. Single quotes<br>'Yes'<br>2. Double quotes<br>"Yes"<br>3. Triple quotes (multi-line)<br>"""Yes<br>We Can"""<br>4. String method<br>str(5) == '5' # True<br>5. Concatenation<br>"Ma" + "hatma" # 'Mahatma'<br><br>These are whitespace characters in strings.<br>● Newline \n<br>● Space \s<br>● Tab \t | ```python
## 4. Indexing and Slicing
s = "The youngest pope was 11 years old"
print(s[0])       # 'T'
print(s[1:3])     # 'he'
print(s[-3:-1])   # 'ol'
print(s[-3:])     # 'old'
x = s.split()     # creates string array of words
print(x[-3] + " " + x[-1] + " " + x[2] + "s")
                  # '11 old popes'

## 5. Most Important String Methods
y = "    This is lazy\t\n    "
print(y.strip()) # Remove Whitespace: 'This is lazy'
print("DrDre".lower()) # Lowercase: 'drdre'
print("attention".upper()) # Uppercase: 'ATTENTION'
print("smartphone".startswith("smart")) # True
print("smartphone".endswith("phone")) # True
print("another".find("other")) # Match index: 2
print("cheat".replace("ch", "m")) # 'meat'
print(','.join(["F", "B", "I"])) # 'F,B,I'
print(len("Rumpelstiltskin")) # String length: 15
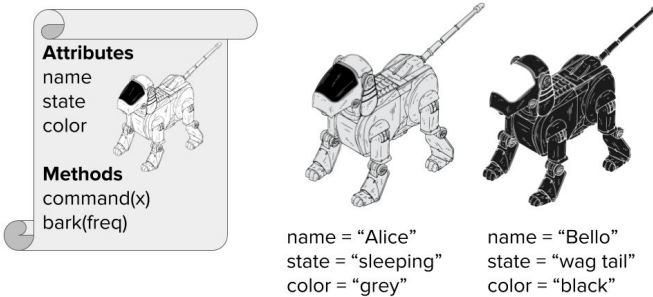print("ear" in "earth") # Contains: True
``` |

finxter

Python Cheat Sheet: Complex Data Types

"A puzzle a day to learn, code, and play" ➜ Visit finxter.com

	Description	Example
List	A container data type that stores a sequence of elements. Unlike strings, lists are mutable: modification possible.	```l = [1, 2, 2]``` ```print(len(l)) # 3```

Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation.
	The append operation is very fast.
		```[1, 2, 2].append(4) # [1, 2, 2, 4]```
		```[1, 2, 4].insert(2,2) # [1, 2, 2, 4]```
		```[1, 2, 2] + [4] # [1, 2, 2, 4]```

Removal	Removing an element can be slower.	```[1, 2, 2, 4].remove(1) # [2, 2, 4]```
Reversing	This reverses the order of list elements.	```[1, 2, 3].reverse() # [3, 2, 1]```
Sorting	Sorts a list. The computational complexity of sorting is linear in the no. list elements.	```[2, 4, 2].sort() # [2, 2, 4]```
Indexing	Finds the first occurence of an element in the list & returns its index. Can be slow as the whole list is traversed.	```[2, 2, 4].index(2) # index of element 4 is "0"``` ```[2, 2, 4].index(2,1) # index of element 2 after pos 1 is "1"```

Stack	Python lists can be used intuitively as stacks via the two list operations append() and pop().	```stack = [3]```
		```stack.append(42) # [3, 42]```
		```stack.pop() # 42 (stack: [3])```
		```stack.pop() # 3 (stack: [])```

Set	A set is an unordered collection of unique elements ("at-most-once").	```basket = {'apple', 'eggs', 'banana', 'orange'}``` ```same = set(['apple', 'eggs', 'banana', 'orange'])```
Dictionary	The dictionary is a useful data structure for storing (key, value) pairs.	```calories = {'apple' : 52, 'banana' : 89, 'choco' : 546}```

Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the keys() and values() functions to access all keys and values of the dictionary.	```print(calories['apple'] < calories['choco']) # True```
		```calories['cappu'] = 74```
		```print(calories['banana'] < calories['cappu']) # False```
		```print('apple' in calories.keys()) # True```
		```print(52 in calories.values()) # True```

Dictionary Looping	You can access the (key, value) pairs of a dictionary with the `items()` method.	```for k, v in calories.items():``` ``` print(k) if v > 500 else None # 'chocolate'```

Membership operator	Check with the 'in' keyword whether the set, list, or dictionary contains an element. Set containment is faster than list containment.	```basket = {'apple', 'eggs', 'banana', 'orange'}```
		```print('eggs' in basket) # True```
		```print('mushroom' in basket) # False```

List and Set Comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses.
	
	Set comprehension is similar to list comprehension.
		```# List comprehension```
		```l = [('Hi ' + x) for x in ['Alice', 'Bob', 'Pete']]```
		```print(l) # ['Hi Alice', 'Hi Bob', 'Hi Pete']```
		```l2 = [x * y for x in range(3) for y in range(3) if x>y]```
		```print(l2) # [0, 0, 2]```
		```# Set comprehension```
		```squares = { x**2 for x in [0,2,4] if x < 4 } # {0, 4}```

finxter

Python Cheat Sheet: Classes

"A puzzle a day to learn, code, and play" ➜ Visit finxter.com

	Description	Example

Classes	A class encapsulates data and functionality: data as attributes, and functionality as methods. It is a blueprint for creating concrete instances in memory.	```python
		class Dog:
		    """ Blueprint of a dog """
		
		    # class variable shared by all instances
		    species = ["canis lupus"]
		
		    def __init__(self, name, color):
		        self.name = name
		        self.state = "sleeping"
		        self.color = color
		
		    def command(self, x):
		        if x == self.name:
		            self.bark(2)
		        elif x == "sit":
		            self.state = "sit"
		        else:
		            self.state = "wag tail"
		
		    def bark(self, freq):
		        for i in range(freq):
		            print("[" + self.name
		                + "]: Woof!")```

For the example code:

```python
class Dog:
    """ Blueprint of a dog """

    # class variable shared by all instances
    species = ["canis lupus"]

    def __init__(self, name, color):
        self.name = name
        self.state = "sleeping"
        self.color = color

    def command(self, x):
        if x == self.name:
            self.bark(2)
        elif x == "sit":
            self.state = "sit"
        else:
            self.state = "wag tail"

    def bark(self, freq):
        for i in range(freq):
            print("[" + self.name
                + "]: Woof!")
```

Class

Instances

Attributes
name
state
color

Methods
command(x)
bark(freq)

name = "Alice"
state = "sleeping"
color = "grey"

name = "Bello"
state = "wag tail"
color = "black"

Instance

You are an instance of the class human. An instance is a concrete implementation of a class: all attributes of an instance have a fixed value. Your hair is blond, brown, or black--but never unspecified.

Each instance has its own attributes independent of other instances. Yet, class variables are different. These are data values associated with the class, not the instances. Hence, all instance share the same class variable `species` in the example.

Self

The first argument when defining any method is always the `self` argument. This argument specifies the instance on which you call the method.

`self` gives the Python interpreter the information about the concrete instance. To *define* a method, you use `self` to modify the instance attributes. But to *call* an instance method, you do not need to specify `self`.

Creation

You can create classes "on the fly" and use them as logical units to store complex data types.

```python
class Employee():
        pass
employee = Employee()
employee.salary = 122000
employee.firstname = "alice"
employee.lastname = "wonderland"

print(employee.firstname + " "
        + employee.lastname + " "
        + str(employee.salary) + "$")
# alice wonderland 122000$
```

Example code continued:

```python
bello = Dog("bello", "black")
alice = Dog("alice", "white")

print(bello.color) # black
print(alice.color) # white

bello.bark(1) # [bello]: Woof!

alice.command("sit")
print("[alice]: " + alice.state)
# [alice]: sit

bello.command("no")
print("[bello]: " + bello.state)
# [bello]: wag tail

alice.command("alice")
# [alice]: Woof!
# [alice]: Woof!

bello.species += ["wulf"]
print(len(bello.species)
        == len(alice.species)) # True (!)
```

Python Cheat Sheet: Functions and Tricks

"A puzzle a day to learn, code, and play" ➜ Visit finxter.com

		Description	Example	Result
A D V A N C E D F U N C T I O N S	`map(func, iter)`	Executes the function on all elements of the iterable	`list(map(lambda x: x[0], ['red', 'green', 'blue']))`	`['r', 'g', 'b']`
	`map(func, i1, ..., ik)`	Executes the function on all k elements of the k iterables	`list(map(lambda x, y: str(x) + ' ' + y + 's' , [0, 2, 2], ['apple', 'orange', 'banana']))`	`['0 apples', '2 oranges', '2 bananas']`
	`string.join(iter)`	Concatenates iterable elements separated by **string**	`' marries '.join(list(['Alice', 'Bob']))`	`'Alice marries Bob'`
	`filter(func, iterable)`	Filters out elements in iterable for which function returns **False** (or 0)	`list(filter(lambda x: True if x>17 else False, [1, 15, 17, 18]))`	`[18]`
	`string.strip()`	Removes leading and trailing whitespaces of string	`print("\n  \t  42  \t ".strip())`	`42`
	`sorted(iter)`	Sorts iterable in ascending order	`sorted([8, 3, 2, 42, 5])`	`[2, 3, 5, 8, 42]`
	`sorted(iter, key=key)`	Sorts according to the key function in ascending order	`sorted([8, 3, 2, 42, 5], key=lambda x: 0 if x==42 else x)`	`[42, 2, 3, 5, 8]`
	`help(func)`	Returns documentation of **func**	`help(str.upper())`	`'... to uppercase.'`
	`zip(i1, i2, ...)`	Groups the i-th elements of iterators i1, i2, ... together	`list(zip(['Alice', 'Anna'], ['Bob', 'Jon', 'Frank']))`	`[('Alice', 'Bob'), ('Anna', 'Jon')]`
	Unzip	Equal to: 1) unpack the zipped list, 2) zip the result	`list(zip(*[('Alice', 'Bob'), ('Anna', 'Jon')]))`	`[('Alice', 'Anna'), ('Bob', 'Jon')]`
	`enumerate(iter)`	Assigns a counter value to each element of the iterable	`list(enumerate(['Alice', 'Bob', 'Jon']))`	`[(0, 'Alice'), (1, 'Bob'), (2, 'Jon')]`
T R I C K S	python -m http.server <P>	Want to share files between PC and phone? Run this command in PC's shell. <P> is any port number 0–65535. Type < IP address of PC>:<P> in the phone's browser. You can now browse the files in the PC directory.		
	Read comic	`import antigravity`	Open the comic series xkcd in your web browser	
	Zen of Python	`import this`	`'...Beautiful is better than ugly. Explicit is ...'`	
	Swapping numbers	Swapping variables is a breeze in Python. No offense, Java!	`a, b = 'Jane', 'Alice'` `a, b = b, a`	`a = 'Alice'` `b = 'Jane'`

	Unpacking arguments	Use a sequence as function arguments via asterisk operator *. Use a dictionary (key, value) via double asterisk operator **	`def f(x, y, z): return x + y * z`
			`f(*[1, 3, 4])`
			`f(**{'z' : 4, 'x' : 1, 'y' : 3})`
				`13`
				`13`

	Extended Unpacking	Use unpacking for multiple assignment feature in Python	`a, *b = [1, 2, 3, 4, 5]`	`a = 1` `b = [2, 3, 4, 5]`

	Merge two dictionaries	Use unpacking to merge two dictionaries into a single one	`x={'Alice' : 18}`
			`y={'Bob' : 27, 'Ann' : 22}`
			`z = {**x,**y}`
				`z = {'Alice': 18, 'Bob': 27, 'Ann': 22}`

f i n x t e r