

Python Cheat Sheet: Keywords

“A puzzle a day to learn, code, and play” → Visit finxter.com

Keyword	Description	Code example
<code>False, True</code>	Data values from the data type Boolean	<code>False == (1 > 2), True == (2 > 1)</code>
<code>and, or, not</code>	Logical operators: (<code>x and y</code>) → both x and y must be True (<code>x or y</code>) → either x or y must be True (<code>not x</code>) → x must be false	<pre>x, y = True, False (x or y) == True # True (x and y) == False # True (not y) == True # True</pre>
<code>break</code>	Ends loop prematurely	<pre>while(True): break # no infinite loop print("hello world")</pre>
<code>continue</code>	Finishes current loop iteration	<pre>while(True): continue print("43") # dead code</pre>
<code>class</code> <code>def</code>	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.	<pre>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</pre>
<code>if, elif, else</code>	Conditional program execution: program starts with “if” branch, tries the “elif” branches, and finishes with “else” branch (until one branch evaluates to True).	<pre>x = int(input("your value: ")) if x > 3: print("Big") elif x == 3: print("Medium") else: print("Small")</pre>
<code>for, while</code>	<pre># For loop declaration for i in [0,1,2]: print(i)</pre>	<pre># While loop - same semantics j = 0 while j < 3: print(j) j = j + 1</pre>
<code>in</code>	Checks whether element is in sequence	<code>42 in [2, 39, 42] # True</code>
<code>is</code>	Checks whether both elements point to the same object	<pre>y = x = 3 x is y # True [3] is [3] # False</pre>
<code>None</code>	Empty value constant	<pre>def f(): x = 2 f() is None # True</pre>
<code>lambda</code>	Function with no name (anonymous function)	<code>(lambda x: x + 3)(3) # returns 6</code>
<code>return</code>	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.	<pre>def incrementor(x): return x + 1 incrementor(4) # returns 5</pre>

Python Cheat Sheet: Basic Data Types

“A puzzle a day to learn, code, and play” → Visit finxter.com

	Description	Example
Boolean	<p>The Boolean data type is a truth value, either True or False.</p> <p>The Boolean operators ordered by priority: not x → “if x is False, then x, else y” x and y → “if x is False, then x, else y” x or y → “if x is False, then y, else x”</p> <p>These comparison operators evaluate to True: 1 < 2 and 0 <= 1 and 3 > 2 and 2 >= 2 and 1 == 1 and 1 != 0 # True</p>	<pre>## 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</pre>
Integer, Float	<p>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).</p> <p>The // operator performs integer division. The result is an integer value that is rounded toward the smaller integer number (e.g. 3 // 2 == 1).</p>	<pre>## 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</pre>
String	<p>Python Strings are sequences of characters.</p> <p>The four main ways to create strings are the following.</p> <ol style="list-style-type: none">1. Single quotes 'Yes'2. Double quotes "Yes"3. Triple quotes (multi-line) """Yes We Can"""4. String method str(5) == '5' # True5. Concatenation "Ma" + "hatma" # 'Mahatma' <p>These are whitespace characters in strings.</p> <ul style="list-style-type: none">• Newline \n• Space \s• Tab \t	<pre>## 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</pre>

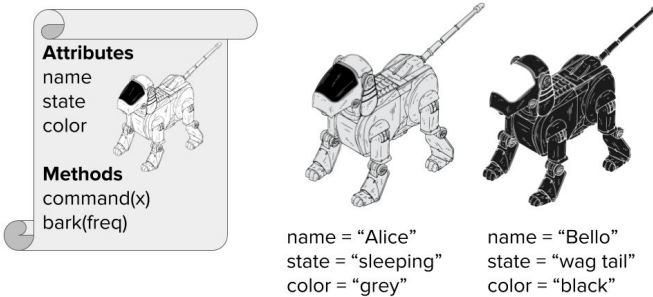
Python Cheat Sheet: Complex Data Types

“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.	<pre>l = [1, 2, 2] print(len(l)) # 3</pre>
Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation. The append operation is very fast.	<pre>[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]</pre>
Removal	Removing an element can be slower.	<pre>[1, 2, 2, 4].remove(1) # [2, 2, 4]</pre>
Reversing	This reverses the order of list elements.	<pre>[1, 2, 3].reverse() # [3, 2, 1]</pre>
Sorting	Sorts a list. The computational complexity of sorting is linear in the no. list elements.	<pre>[2, 4, 2].sort() # [2, 2, 4]</pre>
Indexing	Finds the first occurrence of an element in the list & returns its index. Can be slow as the whole list is traversed.	<pre>[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"</pre>
Stack	Python lists can be used intuitively as stacks via the two list operations append() and pop().	<pre>stack = [3] stack.append(42) # [3, 42] stack.pop() # 42 (stack: [3]) stack.pop() # 3 (stack: [])</pre>
Set	A set is an unordered collection of unique elements (“at-most-once”).	<pre>basket = {'apple', 'eggs', 'banana', 'orange'} same = set(['apple', 'eggs', 'banana', 'orange'])</pre>
Dictionary	The dictionary is a useful data structure for storing (key, value) pairs.	<pre>calories = {'apple' : 52, 'banana' : 89, 'choco' : 546}</pre>
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.	<pre>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</pre>
Dictionary Looping	You can access the (key, value) pairs of a dictionary with the items() method.	<pre>for k, v in calories.items(): print(k) if v > 500 else None # 'chocolate'</pre>
Membership operator	Check with the ‘in’ keyword whether the set, list, or dictionary contains an element. Set containment is faster than list containment.	<pre>basket = {'apple', 'eggs', 'banana', 'orange'} print('eggs' in basket) # True print('mushroom' in basket) # False</pre>
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.	<pre># 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}</pre>

Python Cheat Sheet: Classes

“A puzzle a day to learn, code, and play” → Visit finxter.com

	Description	Example
Classes	<p>A class encapsulates data and functionality: data as attributes, and functionality as methods. It is a blueprint for creating concrete instances in memory.</p> <p>Class Instances</p> 	<pre>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!") 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 (!)</pre>
Instance	<p>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.</p> <p>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.</p>	
Self	<p>The first argument when defining any method is always the self argument. This argument specifies the instance on which you call the method.</p> <p>self gives the Python interpreter the information about the concrete instance. To <i>define</i> a method, you use self to modify the instance attributes. But to <i>call</i> an instance method, you do not need to specify self.</p>	
Creation	<p>You can create classes “on the fly” and use them as logical units to store complex data types.</p> <pre>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\$</pre>	

Python Cheat Sheet: Functions and Tricks

“A puzzle a day to learn, code, and play” → Visit finxter.com

		Description	Example	Result
ADVANCED FUNCTIONS	<code>map(func, iter)</code>	Executes the function on all elements of the iterable	<code>list(map(lambda x: x[0], ['red', 'green', 'blue']))</code>	<code>['r', 'g', 'b']</code>
	<code>map(func, i1, ..., ik)</code>	Executes the function on all k elements of the k iterables	<code>list(map(lambda x, y: str(x) + ' ' + y + 's', [0, 2, 2], ['apple', 'orange', 'banana']))</code>	<code>['0 apples', '2 oranges', '2 bananas']</code>
	<code>string.join(iter)</code>	Concatenates iterable elements separated by string	<code>'marries'.join(list(['Alice', 'Bob']))</code>	<code>'Alice marries Bob'</code>
	<code>filter(func, iterable)</code>	Filters out elements in iterable for which function returns False (or 0)	<code>list(filter(lambda x: True if x>17 else False, [1, 15, 17, 18]))</code>	<code>[18]</code>
	<code>string.strip()</code>	Removes leading and trailing whitespaces of string	<code>print("\n\t42\t".strip())</code>	<code>42</code>
	<code>sorted(iter)</code>	Sorts iterable in ascending order	<code>sorted([8, 3, 2, 42, 5])</code>	<code>[2, 3, 5, 8, 42]</code>
	<code>sorted(iter, key=key)</code>	Sorts according to the key function in ascending order	<code>sorted([8, 3, 2, 42, 5], key=lambda x: 0 if x==42 else x)</code>	<code>[42, 2, 3, 5, 8]</code>
	<code>help(func)</code>	Returns documentation of func	<code>help(str.upper())</code>	<code>'... to uppercase.'</code>
	<code>zip(i1, i2, ...)</code>	Groups the i-th elements of iterators i1, i2, ... together	<code>list(zip(['Alice', 'Anna'], ['Bob', 'Jon', 'Frank']))</code>	<code>[('Alice', 'Bob'), ('Anna', 'Jon')]</code>
	Unzip	Equal to: 1) unpack the zipped list, 2) zip the result	<code>list(zip(*(['Alice', 'Bob'], ('Anna', 'Jon'))))</code>	<code>[('Alice', 'Anna'), ('Bob', 'Jon')]</code>
	<code>enumerate(iter)</code>	Assigns a counter value to each element of the iterable	<code>list(enumerate(['Alice', 'Bob', 'Jon']))</code>	<code>[(0, 'Alice'), (1, 'Bob'), (2, 'Jon')]</code>
TRICKS	<code>python -m http.server <P></code>	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	<code>import antigravity</code>	Open the comic series xkcd in your web browser	
	Zen of Python	<code>import this</code>	<code>'...Beautiful is better than ugly. Explicit is ...'</code>	
	Swapping numbers	Swapping variables is a breeze in Python. No offense, Java!	<code>a, b = 'Jane', 'Alice'</code> <code>a, b = b, a</code>	<code>a = 'Alice'</code> <code>b = 'Jane'</code>
	Unpacking arguments	Use a sequence as function arguments via asterisk operator *. Use a dictionary (key, value) via double asterisk operator **	<code>def f(x, y, z): return x + y * z</code> <code>f(*[1, 3, 4])</code> <code>f(**{'z': 4, 'x': 1, 'y': 3})</code>	<code>13</code> <code>13</code>
	Extended Unpacking	Use unpacking for multiple assignment feature in Python	<code>a, *b = [1, 2, 3, 4, 5]</code>	<code>a = 1</code> <code>b = [2, 3, 4, 5]</code>
	Merge two dictionaries	Use unpacking to merge two dictionaries into a single one	<code>x={'Alice': 18}</code> <code>y={'Bob': 27, 'Ann': 22}</code> <code>z = {**x,**y}</code>	<code>z = {'Alice': 18, 'Bob': 27, 'Ann': 22}</code>

Python Cheat Sheet: 14 Interview Questions

“A puzzle a day to learn, code, and play” → Visit finxter.com

Question	Code	Question	Code
Check if list contains integer x	<pre>l = [3, 3, 4, 5, 2, 111, 5] print(111 in l) # True</pre>	Get missing number in [1...100]	<pre>def get_missing_number(lst): return set(range(lst[len(lst)-1])[1:]) - set(l) l = list(range(1,100)) l.remove(50) print(get_missing_number(l)) # 50</pre>
Find duplicate number in integer list	<pre>def find_duplicates(elements): duplicates, seen = set(), set() for element in elements: if element in seen: duplicates.add(element) seen.add(element) return list(duplicates)</pre>	Compute the intersection of two lists	<pre>def intersect(lst1, lst2): res, lst2_copy = [], lst2[:] for el in lst1: if el in lst2_copy: res.append(el) lst2_copy.remove(el) return res</pre>
Check if two strings are anagrams	<pre>def is_anagram(s1, s2): return set(s1) == set(s2) print(is_anagram("elvis", "lives")) # True</pre>	Find max and min in unsorted list	<pre>l = [4, 3, 6, 3, 4, 888, 1, -11, 22, 3] print(max(l)) # 888 print(min(l)) # -11</pre>
Remove all duplicates from list	<pre>lst = list(range(10)) + list(range(10)) lst = list(set(lst)) print(lst) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]</pre>	Reverse string using recursion	<pre>def reverse(string): if len(string)<=1: return string return reverse(string[1:])+string[0] print(reverse("hello")) # olleh</pre>
Find pairs of integers in list so that their sum is equal to integer x	<pre>def find_pairs(l, x): pairs = [] for (i, el_1) in enumerate(l): for (j, el_2) in enumerate(l[i+1:]): if el_1 + el_2 == x: pairs.append((el_1, el_2)) return pairs</pre>	Compute the first n Fibonacci numbers	<pre>a, b = 0, 1 n = 10 for i in range(n): print(b) a, b = b, a+b # 1, 1, 2, 3, 5, 8, ...</pre>
Check if a string is a palindrome	<pre>def is_palindrome(phrase): return phrase == phrase[::-1] print(is_palindrome("anna")) # True</pre>	Sort list with Quicksort algorithm	<pre>def qsort(L): if L == []: return [] return qsort([x for x in L[1:] if x< L[0]]) + L[0:1] + qsort([x for x in L[1:] if x>=L[0]]) lst = [44, 33, 22, 5, 77, 55, 999] print(qsort(lst)) # [5, 22, 33, 44, 55, 77, 999]</pre>
Use list as stack, array, and queue	<pre># as a list ... l = [3, 4] l += [5, 6] # l = [3, 4, 5, 6] # ... as a stack ... l.append(10) # l = [4, 5, 6, 10] l.pop() # l = [4, 5, 6] # ... and as a queue l.insert(0, 5) # l = [5, 4, 5, 6] l.pop() # l = [5, 4, 5]</pre>	Find all permutations of string	<pre>def get_permutations(w): if len(w)<=1: return set(w) smaller = get_permutations(w[1:]) perms = set() for x in smaller: for pos in range(0,len(x)+1): perm = x[:pos] + w[0] + x[pos:] perms.add(perm) return perms print(get_permutations("nan")) # {'nna', 'ann', 'nan'}</pre>