

Keywords

Keyword	Description	Code Examples
<code>False</code> , <code>True</code>	Boolean data type	<code>False == (1 > 2)</code> <code>True == (2 > 1)</code>
<code>and</code> , <code>or</code> , <code>not</code>	Logical operators → Both are true → Either is true → Flips Boolean	<code>True and True</code> # True <code>True or False</code> # True <code>not False</code> # True
<code>break</code>	Ends loop prematurely	<code>while True:</code> <code>break</code> # finite loop
<code>continue</code>	Finishes current loop iteration	<code>while True:</code> <code>continue</code> <code>print("42")</code> # dead code
<code>class</code>	Defines new class	<code>class Coffee:</code> # Define your class
<code>def</code>	Defines a new function or class method.	<code>def say_hi():</code> <code>print('hi')</code>
<code>if</code> , <code>elif</code> , <code>else</code>	Conditional execution: - "if" condition == True? - "elif" condition == True? - Fallback: else branch	<code>x = int(input("ur val:"))</code> <code>if x > 3: print("Big")</code> <code>elif x == 3: print("3")</code> <code>else: print("Small")</code>
<code>for</code> , <code>while</code>	# For loop <code>for i in [0,1,2]:</code> <code>print(i)</code>	# While loop does same <code>j = 0</code> <code>while j < 3:</code> <code>print(j); j = j + 1</code>
<code>in</code>	Sequence membership	<code>42 in [2, 39, 42]</code> # True
<code>is</code>	Same object memory location	<code>y = x = 3</code> <code>x is y</code> # True <code>[3] is [3]</code> # False
<code>None</code>	Empty value constant	<code>print()</code> is None # True
<code>lambda</code>	Anonymous function	<code>(lambda x: x+3)(3)</code> # 6
<code>return</code>	Terminates function. Optional return value defines function result.	<code>def increment(x):</code> <code>return x + 1</code> <code>increment(4)</code> # returns 5

Basic Data Structures

Type	Description	Code Examples
Boolean	The Boolean data type is either <code>True</code> or <code>False</code> . Boolean operators are ordered by priority: <code>not</code> → <code>and</code> → <code>or</code> <code>{}</code> → <code>{1, 2, 3}</code> →	<code>## Evaluates to True:</code> <code>1<2 and 0<=1 and 3>2 and 2>=2 and 1==1 and 1!=0</code> <code>## Evaluates to False:</code> <code>bool(None or 0 or 0.0 or '' or [] or {} or set())</code> Rule: <code>None</code> , <code>0</code> , <code>0.0</code> , empty strings, or empty container types evaluate to <code>False</code>
Integer, Float	An integer is a positive or negative number without decimal point such as 3. A float is a positive or negative number with floating point precision such as 3.1415926. Integer division rounds toward the smaller integer (example: <code>3//2==1</code>).	<code>## Arithmetic Operations</code> <code>x, y = 3, 2</code> <code>print(x + y)</code> # 5 <code>print(x - y)</code> # 1 <code>print(x * y)</code> # 6 <code>print(x / y)</code> # 1.5 <code>print(x // y)</code> # 1 <code>print(x % y)</code> # 1 <code>print(-x)</code> # -3 <code>print(abs(-x))</code> # 3 <code>print(int(3.9))</code> # 3 <code>print(float(3))</code> # 3.0 <code>print(x ** y)</code> # 9
String	Python Strings are sequences of characters. String Creation Methods: 1. Single quotes <code>>>> 'Yes'</code> 2. Double quotes <code>>>> "Yes"</code> 3. Triple quotes (multi-line) <code>>>> """Yes</code> We Can""" 4. String method <code>>>> str(5) == '5'</code> True 5. Concatenation <code>>>> "Ma" + "hatma"</code> 'Mahatma' Whitespace chars: Newline \n, Space \s, Tab \t	<code>## Indexing and Slicing</code> <code>s = "The youngest pope was 11 years"</code> <code>s[0]</code> # 'T' <code>s[1:3]</code> # 'he' <code>s[-3:-1]</code> # 'ar' <code>s[-3:]</code> # 'ars' <code>x = s.split()</code> <code>x[-2] + " " + x[2] + "s" # '11 popes'</code> <code>## String Methods</code> <code>y = " Hello world\t\n "</code> <code>y.strip()</code> # Remove Whitespace <code>"HI".lower()</code> # Lowercase: 'hi' <code>"hi".upper()</code> # Uppercase: 'HI' <code>"hello".startswith("he")</code> # True <code>"hello".endswith("lo")</code> # True <code>"hello".find("ll")</code> # Match at 2 <code>"cheat".replace("ch", "m")</code> # 'meat' <code>''.join(["F", "B", "I"])</code> # 'FBI' <code>len("hello world")</code> # Length: 15 <code>"ear" in "earth"</code> # True

Complex Data Structures

Type	Description	Example
List	Stores a sequence of elements. Unlike strings, you can modify list objects (they're <i>mutable</i>).	<code>l = [1, 2, 2]</code> <code>print(len(l))</code> # 3
Adding elements	Add elements to a list with (i) <code>append</code> , (ii) <code>insert</code> , or (iii) list concatenation.	<code>[1, 2].append(4)</code> # [1, 2, 4] <code>[1, 4].insert(1,9)</code> # [1, 9, 4] <code>[1, 2] + [4]</code> # [1, 2, 4]
Removal	Slow for lists	<code>[1, 2, 2, 4].remove(1)</code> # [2, 2, 4]
Reversing	Reverses list order	<code>[1, 2, 3].reverse()</code> # [3, 2, 1]
Sorting	Sorts list using fast Timsort	<code>[2, 4, 2].sort()</code> # [2, 2, 4]
Indexing	Finds the first occurrence of an element & returns index. Slow worst case for whole list traversal.	<code>[2, 2, 4].index(2)</code> # index of item 2 is 0 <code>[2, 2, 4].index(2,1)</code> # index of item 2 after pos 1 is 1
Stack	Use Python lists via the list operations <code>append()</code> and <code>pop()</code>	<code>stack = [3]</code> <code>stack.append(42)</code> # [3, 42] <code>stack.pop()</code> # 42 (stack: [3]) <code>stack.pop()</code> # 3 (stack: [])
Set	An unordered collection of unique elements (<i>at-most-once</i>) → fast membership <i>O(1)</i>	<code>basket = {'apple', 'eggs', 'banana', 'orange'}</code> <code>same = set(['apple', 'eggs', 'banana', 'orange'])</code>

Type	Description	Example
Dictionary	Useful data structure for storing (key, value) pairs	<code>cal = {'apple': 52, 'banana': 89, 'choco': 546}</code> # calories
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	<code>print(cal['apple'] < cal['choco'])</code> # True <code>cal['cappu'] = 74</code> <code>print(cal['banana'] < cal['cappu'])</code> # False <code>print('apple' in cal.keys())</code> # True <code>print(52 in cal.values())</code> # True
Dictionary Iteration	You can access the (key, value) pairs of a dictionary with the items() method.	<code>for k, v in cal.items():</code> <code>print(k) if v > 500 else ''</code> # 'choco'
Membership operator	Check with the in keyword if set, list, or dictionary contains an element. Set membership is faster than list membership.	<code>basket = {'apple', 'eggs', 'banana', 'orange'}</code> <code>print('eggs' in basket)</code> # True <code>print('mushroom' in basket)</code> # False
List & set comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a <i>for</i> clause. Close with zero or more <i>for</i> or <i>if</i> clauses. Set comprehension works similar to list comprehension.	<code>l = ['hi' + x for x in ['Alice', 'Bob', 'Pete']]</code> # ['Hi Alice', 'Hi Bob', 'Hi Pete'] <code>l2 = [x * y for x in range(3) for y in range(3) if x>y]</code> # [0, 0, 2] <code>squares = {x**2 for x in [0,2,4] if x < 4}</code> # {0, 4}

