



## Keywords

Keyword	Description	Code Examples
<code>False</code> , <code>True</code>	Boolean data type	<code>False == (1 &gt; 2)</code> <code>True == (2 &gt; 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 # True</code> <code>True or False # True</code> <code>not False # True</code>
<code>break</code>	Ends loop prematurely	<code>while True:</code> <code>  break # finite loop</code>
<code>continue</code>	Finishes current loop iteration	<code>while True:</code> <code>  continue</code> <code>  print("42") # dead code</code>
<code>class</code>	Defines new class	<code>class Coffee:</code> <code>  # Define your class</code>
<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 &gt; 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 &lt; 3:</code> <code>  print(j); j = j + 1</code>
<code>in</code>	Sequence membership	<code>42 in [2, 39, 42] # True</code>
<code>is</code>	Same object memory location	<code>y = x = 3</code> <code>x is y # True</code> <code>[3] is [3] # False</code>
<code>None</code>	Empty value constant	<code>print() is None # True</code>
<code>lambda</code>	Anonymous function	<code>(lambda x: x+3)(3) # 6</code>
<code>return</code>	Terminates function. Optional return value defines function result.	<code>def increment(x):</code> <code>  return x + 1</code> <code>increment(4) # returns 5</code>

## 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&lt;2 and 0&lt;=1 and 3&gt;2 and 2&gt;=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>  <b>Rule: None, 0, 0.0, empty strings, or empty container types evaluate to False</b>
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: 3//2==1).	<code>## Arithmetic Operations</code> <code>x, y = 3, 2</code> <code>print(x + y) # = 5</code> <code>print(x - y) # = 1</code> <code>print(x * y) # = 6</code> <code>print(x / y) # = 1.5</code> <code>print(x // y) # = 1</code> <code>print(x % y) # = 1</code> <code>print(-x) # = -3</code> <code>print(abs(-x)) # = 3</code> <code>print(int(3.9)) # = 3</code> <code>print(float(3)) # = 3.0</code> <code>print(x ** y) # = 9</code>
String	Python Strings are sequences of characters.  String Creation Methods: 1. Single quotes >>> "Yes" 2. Double quotes >>> "Yes" 3. Triple quotes (multi-line) >>> """Yes We Can""" 4. String method >>> str(5) == '5' True 5. Concatenation >>> "Ma" + "hatma" '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] # 'T'</code> <code>s[1:3] # 'he'</code> <code>s[-3:-1] # 'ar'</code> <code>s[-3:] # 'ars'</code>  <b>Slice [::2]</b>  <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() # Remove Whitespace</code> <code>"HI".lower() # Lowercase: 'hi'</code> <code>"hi".upper() # Uppercase: 'HI'</code> <code>"hello".startswith("he") # True</code> <code>"hello".endswith("lo") # True</code> <code>"hello".find("ll") # Match at 2</code> <code>"cheat".replace("ch", "m") # 'meat'</code> <code>''.join(["F", "B", "I"]) # 'FBI'</code> <code>len("hello world") # Length: 15</code> <code>"ear" in "earth" # True</code>

## 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)) # 3</code>
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) # [1, 2, 4]</code> <code>[1, 4].insert(1,9) # [1, 9, 4]</code> <code>[1, 2] + [4] # [1, 2, 4]</code>
Removal	Slow for lists	<code>[1, 2, 2, 4].remove(1) # [2, 2, 4]</code>
Reversing	Reverses list order	<code>[1, 2, 3].reverse() # [3, 2, 1]</code>
Sorting	Sorts list using fast Timsort	<code>[2, 4, 2].sort() # [2, 2, 4]</code>
Indexing	Finds the first occurrence of an element & returns index. Slow worst case for whole list traversal.	<code>[2, 2, 4].index(2)</code> <code># index of item 2 is 0</code> <code>[2, 2, 4].index(2,1)</code> <code># index of item 2 after pos 1 is 1</code>
Stack	Use Python lists via the list operations <code>append()</code> and <code>pop()</code>	<code>stack = [3]</code> <code>stack.append(42) # [3, 42]</code> <code>stack.pop() # 42 (stack: [3])</code> <code>stack.pop() # 3 (stack: [])</code>
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} # calories</code>
Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the <code>keys()</code> and <code>values()</code> functions to access all keys and values of the dictionary	<code>print(cal['apple'] &lt; cal['choco']) # True</code> <code>cal['cappu'] = 74</code> <code>print(cal['banana'] &lt; cal['cappu']) # False</code> <code>print('apple' in cal.keys()) # True</code> <code>print(52 in cal.values()) # True</code>
Dictionary Iteration	You can access the (key, value) pairs of a dictionary with the <code>items()</code> method.	<code>for k, v in cal.items():</code> <code>  print(k) if v &gt; 500 else ''</code> <code># 'choco'</code>
Membership operator	Check with the <code>in</code> 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) # True</code> <code>print('mushroom' in basket) # False</code>
List & set comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a <code>for</code> clause. Close with zero or more <code>for</code> or <code>if</code> clauses. Set comprehension works similar to list comprehension.	<code>l = ['hi' + x for x in ['Alice', 'Bob', 'Pete']]</code> <code># ['Hi Alice', 'Hi Bob', 'Hi Pete']</code> <code>l2 = [x * y for x in range(3) for y in range(3) if x&gt;y] # [0, 0, 2]</code> <code>squares = { x**2 for x in [0,2,4] if x &lt; 4 } # {0, 4}</code>