


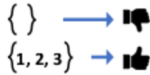



## Keywords

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

## 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 <b>keys()</b> and <b>values()</b> functions to access all keys and values of the dictionary	<code>print(cal['apple'] &lt; cal['choco'])</code> # True <code>cal['cappu'] = 74</code> <code>print(cal['banana'] &lt; 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 <b>items()</b> method.	<code>for k, v in cal.items():</code> <code>print(k) if v &gt; 500 else ''</code> # 'choco'
Membership operator	Check with the <b>in</b> 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 <b>for</b> clause. Close with zero or more <b>for</b> or <b>if</b> 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&gt;y]</code> # [0, 0, 2]  <code>squares = { x**2 for x in [0,2,4] if x &lt; 4 }</code> # {0, 4}

Subscribe to the **11x FREE** Python Cheat Sheet Course:<https://blog.finxter.com/python-cheat-sheets/>