

[Python List Comprehension]

by Alex Kelin

prompt	command	result
Concept	<pre>variable = [**result** for element in entity if condition]</pre>	<p>**entity could be range(), list, any iterable value...</p>
Multiply each number	<pre>lib = [4,8,2,4,0,3] double_nums = [num * 2 for num in lib]</pre>	<pre>>>> print(double_nums) [8, 16, 4, 8, 0, 6]</pre>
POW each number	<pre>lib = [4,8,2,4,0,3] pow_nums = [pow(x, 2) for x in lib]</pre>	<pre>>>> print(pow_nums) [16, 64, 4, 16, 0, 9]</pre>
Reverse a list	<pre>one = ['a', 'b', 'c', 'd', 'e'] two = one[::-1] or three = ['a', 'b', 'c', 'd', 'e'][::-1]</pre>	<pre>>>> print(two) ['e', 'd', 'c', 'b', 'a'] >>> print(three) ['e', 'd', 'c', 'b', 'a']</pre>
Traverse a list	<pre>one = ['a', 'b', 'c', 'd', 'e', 'f', 'g'] result = one[2:6:2] or result = [x for x in one[2:6:2]]</pre>	<pre>>>> print(result) ['c', 'e']</pre>
Operations with strings	<pre>names = ['Bob', 'Mike', 'John'] new_list = ["Hi, " + name for name in names] or new_list = [f 'Hi, {name}' for name in names]</pre>	<pre>>>> print(new_list) ['Hi, Bob', 'Hi, Mike', 'Hi, John']</pre>
String call of the first char	<pre>names = ['Bob', 'Mike', 'John', 'Jerry'] new_list = [x[0] for x in names]</pre>	<pre>>>> print(new_list) ['B', 'M', 'J', 'J']</pre>
Length of a string	<pre>names = ['Bob', 'Mike', 'John', 'Jerry'] lengths = [len(x) for x in names]</pre>	<pre>>>> print(lengths) [3, 4, 4, 5]</pre>
Unique values only	<pre>values = ['h',1,'b','b',4,'1','a',4] option_1 = list({x for x in values}) or option_2 = list(set(values)) or option_3 = [x for x in set(values)] or option_4 = [] [option_4.append(x) for x in values if x not in option_4]</pre>	<pre>>>> print(option_1) [1, 'h', 4, 'a', 'b', '1'] >>> print(option_2) [1, 'h', 4, 'a', 'b', '1'] >>> print(option_3) [1, 'h', 4, 'a', 'b', '1'] >>> print(option_4) ['h', 1, 'b', 4, '1', 'a']</pre>
Common values	<pre>one = ['a', 1, 'b', 'b', 4, '1'] two = ['h', '1', 1, 'a', 'j', '1'] common = [x for x in one if x in two]</pre>	<pre>>>> print(common) ['a', 1, '1']</pre>
Unite two lists	<pre>a = [5,1,6] b = [3,2,4] united = [x for y in [a, b] for x in y] or united = [x for x in a + b]</pre>	<pre>>>> print(united) [5, 1, 6, 3, 2, 4]</pre>

Create nested list	<pre> one = ['Jack', 'Brit', 'Lucas', 'Ben'] two = [10, 15, 4, 6] nl = [[name, age] for name, age in zip(one, two)] or nl = [[one[i], two[i]] for i in range(len(one))] </pre>	<pre> >>> print(nl) [['Jack', 10], ['Brit', 15], ['Lucas', 4], ['Ben', 6]] </pre>
Nested list sum	<pre> nl = [[4, 8], [15, 2], [23, 42]] sum = [x + y for x, y in nl] or sum = [x + y for (x, y) in nl] </pre>	<pre> >>> print(sum) [12, 31, 65] </pre>
Nested list check	<pre> nl = [[4, 8], [15, 2], [23, 42]] check = [x > y for x, y in nl] or check = [x > y for (x, y) in nl] </pre>	<pre> >>> print(check) [False, True, False] </pre>
Sum integers two lists	<pre> a = [5, 1, 6] b = [3, 2, 4] new = [x + y for x, y in zip(a, b)] </pre>	<pre> >>> print(new) [8, 3, 10] </pre>
Conditional comprehension I, ternery operator	<pre> one = [1, 2, 3, 4, 5, 6, 7] new = [x if x % 2 == 0 else x * 2 for x in one] </pre>	<pre> >>> print(new) [2, 2, 6, 4, 10, 6, 14] </pre>
Conditional comprehension II	<pre> a = [1, 2, 3, 4, 5, 6, 7, 8, 9] b = [x for x in a if x > 5 and x % 2 == 0] </pre>	<pre> >>> print(b) [6, 8] </pre>
Multiple Condition comprehension I	<pre> sent = 'it is I, Kai, Jack, and Brit' c = [x for x in sent.split() if x[0].isupper() and len(x) > 1 if ',' not in x] </pre>	<pre> >>> print(c) ['Brit'] </pre>
Multiple Condition comprehension II	<pre> all_clients = [{'name': 'Jack', 'age': 10, 'balance': 100}, {'name': 'Brit', 'age': 15, 'balance': 200}, {'name': 'Lucas', 'age': 4, 'balance': 300}, {'name': 'Ben', 'age': 6, 'balance': 400}] checked = [x['name'] for x in all_clients if x['balance'] >= 300 or x['age'] > 20] </pre>	<pre> >>> print(checked) ['Lucas', 'Ben'] </pre>
Opposite boolean	<pre> booleans = [True, False, True] result = [not x for x in booleans] </pre>	<pre> >>> print(result) [False, True, False] </pre>
Check for value I	<pre> names = ['Bob', 'Mike', 'John', 'Jerry'] check = [x == 'John' for x in names] </pre>	<pre> >>> print(check) [False, False, True, False] </pre>
Check for value II	<pre> lib = [4, 8, 2, 4] check = [x > 3 for x in lib] </pre>	<pre> >>> print(check) [True, True, False, True] </pre>
Search for value index	<pre> names = ['Bob', 'Mike', 'John', 'Jerry', 'John'] check = [i for i, x in enumerate(names) if x == 'John'] </pre>	<pre> >>> print(check) [2, 4] </pre>

{ Python : Dictionary Comprehension }

by Alex Kelin

prompt	command	result
Concept	<pre>dictionary = { key: value for vars in iterable if condition}</pre>	<pre>While comprehension len(keys)=len(values)</pre>
Merge two lists	<pre>one = [1, 2, 3, 4, 5] two = ['a', 'b', 'c', 'd', 'e'] hm = dict([(one[i], two[i]) for i in range(len(one))]) or hm = {one[i]: two[i] for i in range(len(one))}</pre>	<pre>>>> print(hm) {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}</pre>
Merge two lists with zip()	<pre>one = [1, 2, 3, 4, 5] two = ['a', 'b', 'c', 'd', 'e'] hm = {key: value for (key, value) in zip(one, two)} or hm = dict(zip(one, two)) or hm = {a: b for a, b in zip(one, two)}</pre>	<pre>>>> print(hm) {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}</pre>
Merge dicts	<pre>one = {1: 'a', 2: 'b', 3: 'c'} two = {4: 'd', 5: 'e', 6: 'f'} united = {**one, **two} or one.update(two) or one.update(two) united = one</pre>	<pre>>>> print(united) {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'} >>> print(one) {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'} >>> print(united) {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'}</pre>
Add values	<pre>one = {1: 'a', 2: 'b', 3: 'c'} one[4] = 'd' one[5] = 'e' one[6] = 'f' or one.update({4: 'd', 5: 'e', 6: 'f'})</pre>	<pre>>>> print(one) {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'}</pre>
Moderate dict	<pre>old_stock = {'water': 1.42, 'cheese': 2.5, 'milk': 2.0} price = 0.76 correction = {item: value*price for (item, value) in old_stock.items()}</pre>	<pre>>>> print(correction) {'water': 1.0792, 'cheese': 1.9, 'milk': 1.52}</pre>
Unique values only (order preserved)	<pre>one = [4, 1, 2, 2, 3, 1] two = ['a', 'a', 'c', 'c', 'e'] new_two = [] uv = [new_two.append(i) for i in two if i not in new_two] result = {i: j for i, j in zip(one, new_two)}</pre>	<pre>>>> print(result) {4: 'a', 1: 'c', 2: 'e'}</pre>
Unique values only (order not preserved)	<pre>one = [4, 1, 2, 2, 3, 1] two = ['a', 'a', 'c', 'c', 'e'] result = {i: j for i, j in zip(one, set(two))}</pre>	<pre>>>> print(result) {4: 'c', 1: 'e', 2: 'a'}</pre>

Limited by values length	<pre> a = [1, 2, 3, 4, 5] b = ['a', 'b', 'c'] result = {k: v for k, v in zip(b, a[:len(b)])} </pre>	<pre> >>> print(result) {'a': 1, 'b': 2, 'c': 3} </pre>
Dict with multiple conditions	<pre> names = {'mike': 10, 'jack': 32, 'rachel': 55} new_dict = {k: v for (k, v) in names.items() if v % 2 == 0 if v > 20} </pre>	<pre> >>> print(new_dict) {'jack': 32} </pre>
Conditional comprehension I	<pre> a = {'mike': 10, 'jack': 32, 'rachel': 55} new_dict = {k: v for (k, v) in a.items() if v % 2 == 0} </pre>	<pre> >>> print(a) {'mike': 10, 'jack': 32} </pre>
Conditional comprehension II, ternery operator	<pre> names = {'jack': 38, 'tina': 48, 'ron': 57, 'john': 33} new_dict = {x: ('old' if y > 40 else 'young') for (x, y) in names.items()} </pre>	<pre> >>> print(new_dict) {'jack': 'young', 'tina': 'old', 'Ron': 'old', 'john': 'young'} </pre>
Conditional comprehension III, ternery operator	<pre> names = ['alice', 'bob', 'kate', 'kimber'] size = [1, 2, 3, 4, 5, 6, 7] view = {names[i].capitalize(): (f' {size[i]} nice' if i >= 2 else f' {size[i]} small') for i in range(len(names))} </pre>	<pre> >>> print(view) {'Alice': ' 1 small', 'Bob': ' 2 small', 'Kate': ' 3 nice', 'Kimber': ' 4 nice'} </pre>
Nested dictionary comprehension I	<pre> keys = ['a', 'b', 'c', 'd'] values = [1, 2, 3] res = {k1: {k2: k1 for k2 in values} for k1 in keys} </pre>	<pre> >>> print(res) {'a': {1: 'a', 2: 'a', 3: 'a'}, 'b': {1: 'b', 2: 'b', 3: 'b'}, 'c': {1: 'c', 2: 'c', 3: 'c'}, 'd': {1: 'd', 2: 'd', 3: 'd'}} </pre>
Nested dictionary comprehension II	<pre> keys = ['a', 'b', 'c', 'd'] values = [1, 2, 3] res = {k1: [x for x in values] for k1 in keys} </pre>	<pre> >>> print(res) {'a': [1, 2, 3], 'b': [1, 2, 3], 'c': [1, 2, 3], 'd': [1, 2, 3]} </pre>
Find length of variable	<pre> names = ['Alex', 'Tom', 'Johnson', 'Bi', 'Foobar'] counted = {x.lower(): len(x) for x in names if x} </pre>	<pre> >>> print(counted) {'alex': 4, 'tom': 3, 'johnson': 7, 'bi': 2, 'foobar': 6} </pre>

Python lambda λ : functions

by Alex Kelin

prompt	command	result
Concept	<code>function_name = (lambda variable(s): result if condition else result_2)</code>	Condition and ternery is optional
Simple value operation	<code>squared = lambda x: x ** 2</code>	<pre>>>> print(squared(5)) 25</pre>
Multiple value operation	<code>value = lambda x, y: x + 2 - y</code>	<pre>>>> print(value(2,1)) 3</pre>
Check single value	<code>check_num = (lambda x: f'{x} is greater than 5' if x > 5 else f'{x} is not greater than 5')</code>	<pre>>>> print(check_num(6)) 6 is greater than 5</pre>
Check multiple values	<code>check_num = (lambda x, y: f'{x} and {y} are greater than 5' if x > 5 and y > 5 else f'{x} and {y} are not greater than 5')</code>	<pre>>>> print(check_num(2,3)) 2 and 3 are not greater than 5</pre>
Check for value	<code>lib = ['a', 'b', 'c', 'd'] boolean = list(map(lambda x: x == 'b', lib))</code>	<pre>>>> print(boolean) [False, True, False, False]</pre>
Any() or all() value check	<code>lst = [1, 2, 3, 4, 5] check_1 = any(map(lambda x: x % 2 == 0, lst)) check_2 = all(map(lambda x: x % 2 == 0, lst))</code>	<pre>>>> print(check_1) True >>> print(check_2) False</pre>
Operations with list	<code>lib = [3, 1, 2] a = [lambda x=_: x + 1 for _ in lib] b = [(lambda x: x * 2)(_()) for _ in a]</code>	<pre>>>> print(a) [<function <listcomp>.<lambda> at 0x104761a80>,] >>> print(b) [8, 4, 6]</pre>
	<code>lib = [3, 1, 2, 4] c = list(map(lambda x: x, lib)) d = list(map(lambda x: x / 2, lib))</code>	<pre>>>> print(c) [3, 1, 2, 4] >>> print(d) [1.5, 0.5, 1.0, 2.0]</pre>
	<code>lib = ['Bob', 'Mike', 'John', 'Jerry'] e = list(map(lambda x: f' Hi, {x}', lib))</code>	<pre>>>> print(e) [' Hi, Bob', ' Hi, Mike', ' Hi, John', ' Hi, Jerry']</pre>
	<code>lib_1 = ['a', 'b', 'c', 'd'] lib_2 = [20, 'M', 'T', 'V'] f = list(map(lambda x, y: f'{x} - {y}', lib_1, lib_2))</code>	<pre>>>> print(f) ['a - 20', 'b - M', 'c - T', 'd - V']</pre>
Operations with dict	<code>a = ['a', 'b', 'c'] b = [1, 2, 3] new_dict = dict(zip(a, map(lambda x: x+1, b)))</code>	<pre>>>> print(new_dict) {'a': 2, 'b': 3, 'c': 4}</pre>
Determine length of a string	<code>names = ['Bob', 'Mike', 'John', 'Jerry'] lengths = [len(x) for x in names]</code>	<pre>>>> print(lengths) [3, 4, 4, 5]</pre>
Opposite boolean	<code>booleans = [True, False, True] result = [not x for x in booleans]</code>	<pre>>>> print(result) [False, True, False]</pre>
Extract positive values	<code>my_list = [1, -2, 3, -4, 5] pos_nums = list(filter(lambda x: x > 0, my_list))</code>	<pre>>>> print(pos_nums) [1, 3, 5]</pre>