[Python List Comprehension]

by Alex Kelin

prompt	command	result
Concept	variable = [***result*** for element in entity if condition]	**entity could be range(), list, any iterable value…
Multiply each number	lib = [4,8,2,4,0,3] double_nums = [num * 2 for num in lib]	>>> print(double_nums) [8, 16, 4, 8, 0, 6]
POW each number	lib = [4,8,2,4,0,3] pow_nums = [pow (x, 2) for x in lib]	>>> print(pow_nums) [16, 64, 4, 16, 0, 9]

Reverse a list	one = ['a', 'b', 'c', 'd', 'e']
	two = one[::-1]
	or
	three = ['a', 'b', 'c', 'd', 'e'][::-1]
		>>> print(two)
		['e', 'd', 'c', 'b', 'a']
		>>> print(three)
		['e', 'd', 'c', 'b', 'a']

Traverse a list: every second value from indexes 1 to 6	one = ['a', 'b', 'c', 'd', 'e', 'f', 'g'] result = one[2:6:2] or result = [x for x in values[2:6:2]]	>>> print(result) ['c', 'e']
Operations with strings	names = ['Bob', 'Mike', 'John'] new_list = ["Hi, " + name for name in names] or new_list = [f 'Hi, {name}' for name in names]	>>> print(new_list) ['Hi, Bob', 'Hi, Mike', 'Hi, John']
String call of the first char	names = ['Bob', 'Mike', 'John', 'Jerry'] new_list = [x[0] for x in names]	>>> print(new_list) ['B', 'M', 'J', 'J']
Length of a string	names = ['Bob', 'Mike', 'John', 'Jerry'] lengths = [len(x) for x in names]	>>> print(lengths) [3, 4, 4, 5]

Unique values only	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]
		>>> 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']

Common values	one = ['a', 1, 'b', 'b', 4, '1'] two = ['h', 'l', 1, 'a', 'j', '1'] common = [x for x in one if x in two]	>>> print(common) ['a', 1, '1']
Unite two lists	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]	>>> print(united) [5, 1, 6, 3, 2, 4]

Create nested list	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))]	>>> print(nl) [['Jack', 10], ['Brit', 15], ['Lucas', 4], ['Ben', 6]]
Nested list sum	nl = [[4, 8], [15, 2], [23, 42]] sum = [x + y for x, y in nl] or sum = [x + y for (x, y) in nl]	>>> print(sum) [12, 31, 65]
Nested list check	nl = [[4, 8], [15, 2], [23, 42]] check = [x > y for x, y in nl] or check = [x > y for (x, y) in nl]	>>> print(check) [False, True, False]
Sum integers two lists	a = [5, 1, 6] b = [3, 2, 4] new = [x + y for x, y in zip(a, b)]	>>> print(new) [8, 3, 10]
Conditional comprehension I, ternery operator	one = [1, 2, 3, 4, 5, 6, 7] new = [x if x % 2 == 0 else x * 2 for x in one]	>>> print(new) [2, 2, 6, 4, 10, 6, 14]
Conditional comprehension II	a = [1, 2, 3, 4, 5, 6, 7, 8, 9] b = [x for x in a if x > 5 and x % 2 == 0]	>>> print(b) [6, 8]
Multiple Condition comprehension I	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]	>>> print(c) [‘Brit']
Multiple Condition comprehension II	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]	>>> print(checked) ['Lucas', 'Ben']
Opposite boolean	booleans = [True, False, True] result = [not x for x in booleans]	>>> print(result) [False, True, False]
Check for value I	names = ['Bob', 'Mike', 'John', 'Jerry'] check = [x == 'John' for x in names]	>>> print(check) [False, False, True, False]
Check for value II	lib = [4, 8, 2, 4] check = [x > 3 for x in lib]	>>> print(check) [True, True, False, True]
Search for value index	names = ['Bob', 'Mike', 'John', 'Jerry', 'John'] check = [i for i, x in enumerate(names) if x == 'John']	>>> print(check) [2, 4]

{ Python : Dictionary Comprehension }

by Alex Kelin

prompt	command	result
Concept	`dictionary = { key: value for vars in iterable if condition}`	While comprehension len(keys)=len(values)

Merge two lists	`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))}`
		`>>> print(hm)`
		`{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}`

Merge two lists with zip()	`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)}`
		`>>> print(hm)`
		`{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}`

Merge dicts	`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`
		`>>> 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'}`

Add values	`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'})`
		`>>> print(one)`
		`{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'}`

Moderate dict	`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()}`
		`>>> print(correction)`
		`{'water': 1.0792, 'cheese': 1.9, 'milk': 1.52}`

Unique values only (order preserved)	`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)}`
		`>>> print(result)`
		`{4: 'a', 1: 'c', 2: 'e'}`

Unique values only (order not preserved)	`one = [4, 1, 2, 2, 3, 1]`
	`two = ['a', 'a', 'c', 'c', 'e']`
	`result = {i: j for i, j in zip(one, set(two))}`
		`>>> print(result)`
		`{4: 'c', 1: 'e', 2: 'a'}`

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

Python lambda λ : functions

by Alex Kelin

prompt	command	result
Concept	`function_name = ( lambda variable(s): result if condition else result_2 )`	Condition and ternery is optional
Simple value operation	`squared = lambda x: x ** 2`	`>>> print(squared(5))` 25
Multiple value operation	`value = lambda x, y: x + 2 - y`	`>>> print(value(2,1))` 3

Check single value	`check_num = (`
	`  lambda x: f'{x} is greater than 5' if x > 5`
	`  else f'{x} is not greater than 5')`
		`>>> print(check_num(6))`
		6 is greater than 5

Check multiple values	`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')`
		`>>> print(check_num(2,3))`
		2 and 3 are not greater than 5

Check for value	`lib = ['a', 'b', 'c', 'd']` `boolean = list(map(lambda x: x == 'b', lib))`	`>>> print(boolean)` [False, True, False, False]

Any() or all() value check	`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))`
		`>>> print(check_1)`
		True
		`>>> print(check_2)`
		False

Operations with list	`lib = [3, 1, 2]`
	`a = [lambda x=_: x + 1 for _ in lib]`
	`b = [(lambda x: x * 2)(_()) for _ in a]`
		`>>> print(a)`
		[<function <listcomp>.<lambda> at 0x10476la80>, ...]
		`>>> print(b)`
		[8, 4, 6]

	`lib = [3, 1, 2, 4]`
	`c = list(map(lambda x: x, lib))`
	`d = list(map(lambda x: x / 2, lib))`
		`>>> print(c)`
		[3, 1, 2, 4]
		`>>> print(d)`
		[1.5, 0.5, 1.0, 2.0]

	`lib = ['Bob', 'Mike', 'John', 'Jerry']` `e = list(map(lambda x: f' Hi, {x}', lib))`	`>>> print(e)` [' Hi, Bob', ' Hi, Mike', ' Hi, John', ' Hi, Jerry']

	`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))`
		`>>> print(f)`
		['a - 20', 'b - M', 'c - T', 'd - V']

Operations with dict	`a = ['a', 'b', 'c']`
	`b = [1, 2, 3]`
	`new_dict = dict(zip(a, map(lambda x: x+1, b)))`
		`>>> print(new_dict)`
		{'a': 2, 'b': 3, 'c': 4}

Determine length of a string	`names = ['Bob', 'Mike', 'John', 'Jerry']` `lengths = [len(x) for x in names]`	`>>> print(lengths)` [3, 4, 4, 5]
Opposite boolean	`booleans = [True, False, True]` `result = [not x for x in booleans]`	`>>> print(result)` [False, True, False]
Extract positive values	`my_list = [1, -2, 3, -4, 5]` `pos_nums = list(filter(lambda x: x > 0, my_list))`	`>>> print(pos_nums)` [1, 3, 5]