JÖNKÖPING UNIVERSITY

School of Engineering

LISTS IN PYTHON

Peter Larsson-Green
Jönköping University
Autumn 2018

LISTS

Used to store multiple values.

- Expressions creating lists:

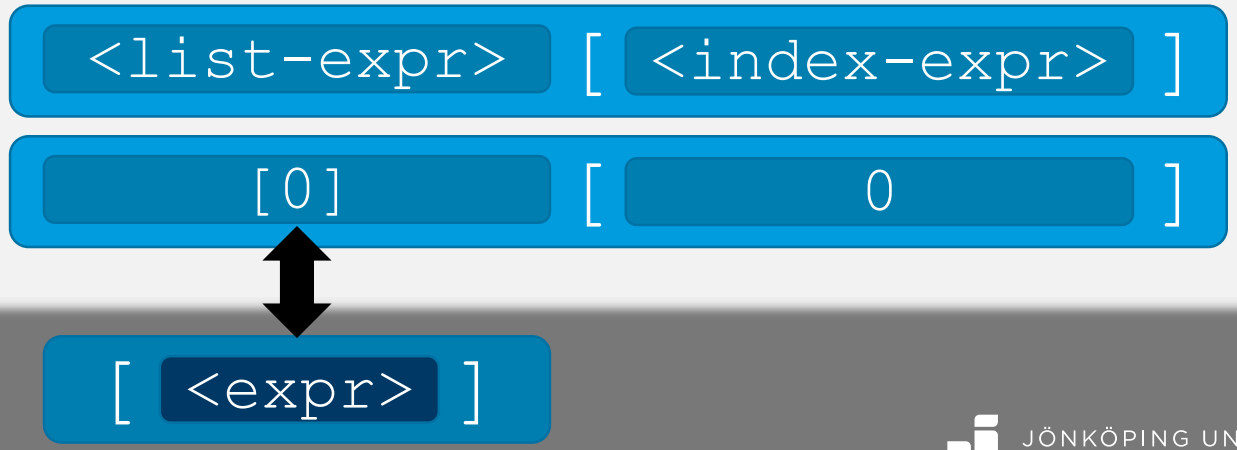
```
[]
[ <expr> ]
[ <expr>, <expr>, ...]
```

- Expression for retrieving an element from the list:

```
<list-expr> [ <index-expr> ]
```

LISTS EXAMPLE

```
my_list = ["a", "b", "c", 4]
x = my_list[1]
y = my_list[0]
z = my_list[my_list[3]-3]
#    my_list[      4     -3]
#    my_list[       1      ]
#               "b"
w = [0][0]
len([3, 7, 4]) → 3
```



EXAMPLE

```
is_5_in_list([3, 5, 8]) → True
is_5_in_list([3, 6, 8]) → False
```

```
def is_5_in_list(the_list):
  i = 0
  while i < len(the_list):
    if the_list[i] == 5:
      return True
    i += 1
  return False
```

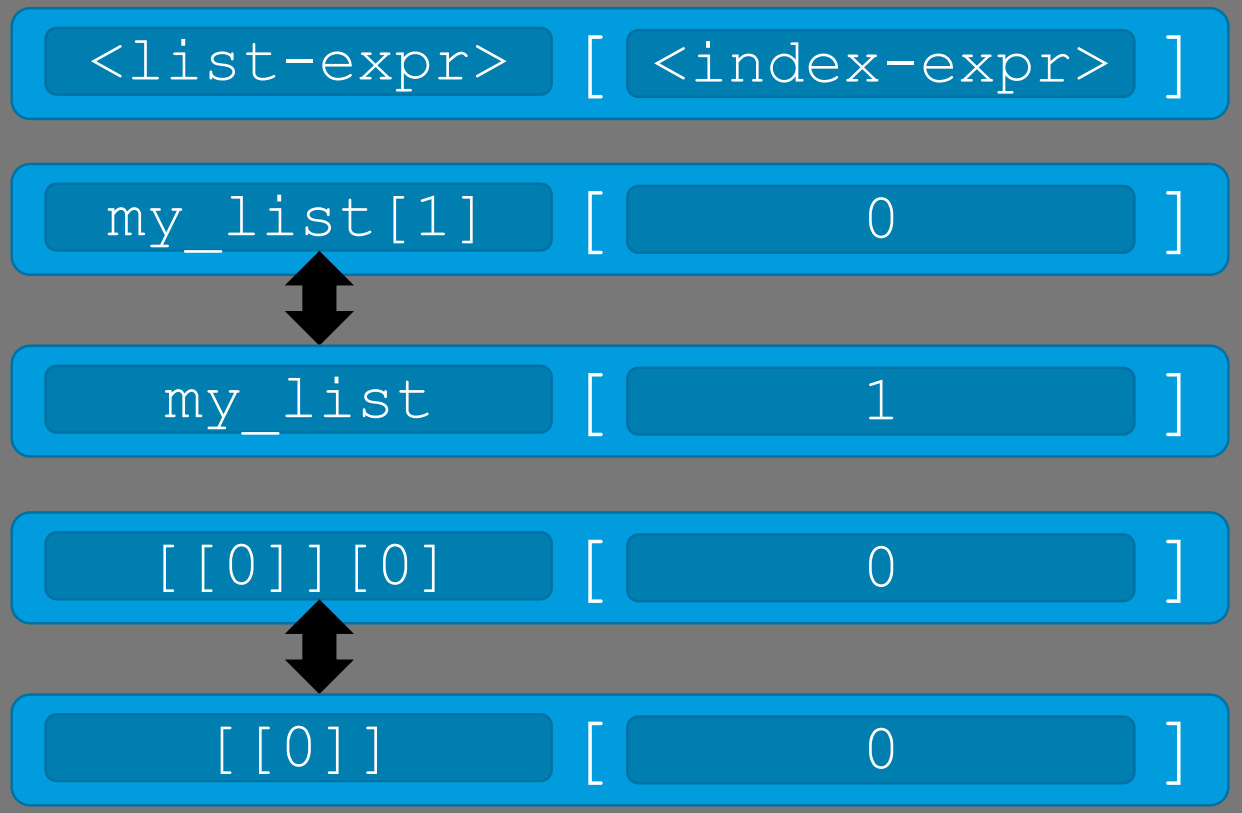
```
def is_5_in_list(the_list):
  for element in the_list:
    if element == 5:
      return True
  return False
```

NESTED LISTS

Lists in lists!

```
my_list = [
   ["a1", "b1", "c1"],
   ["a2", "b2", "c2"]
]
x = my_list[1]
y = x[1]
z = my_list[1][0]
w = [[0]][0][0]
```

```
len([1, [2, 3], 4]) → 3
```



MORE LIST OPERATIONS

- Change an element in a list:
  `the_list[<expr>] = <expr>`
- Add an element to the end of a list:
  `the_list.append(<expr>)`
- Remove an element from a list:
  `the_list.remove(<expr>)`
- Remove the element at a specific index in the list:
  `the_list.pop(<expr>)`
- Documentation:
  `help([])`

LISTS EXAMPLE

```
my_list = ["a", "b", "c"]
my_list[2] = "d"
# my_list = ["a", "b", "d"]
my_list.remove("b")
# my_list = ["a", "d"]
my_list.pop(1)
# my_list = ["a"]
my_list.append("e")
# my_list = ["a", "e"]
```

EXAMPLE

`get_all_plus_1([3, 6, 8])` → `[4, 7, 9]`

```
def get_all_plus_1(the_list):
  new_list = []
  for number in the_list:
    new_list.append(number+1)
  return new_list
```

```
def sum(a_list):
  sum = 0
  for number in a_list:
    sum += number
  return sum
```

VARIABLES STORE REFERENCES

Values in variables aren't copied when used (the reference is!).

```
a = "ab"
b = a
b = "cd"
```

```
a = "ab"
b = a
b = "cd"
```

```
a = [1]
b = a
b.append(2)
# a is [1, 2]!
```

Name	Value
a	ab
b	ab cd

Variable table.

Name	Value
a	→ ab
b	→ cd

Variable table.

Name	Value
a	→ [1, 2]
b	

Variable table.

LISTS EXAMPLE

```
def multiply_by_two(a_list):
  new_list = []
  for number in a_list:
    new_list.append(number*2)
  return new_list

list = [2, 5]
list2 = multiply_by_two(list)
```

```
def multiply_by_two(a_list):
  for i in range(len(a_list)):
    a_list[i] = a_list[i]*2



list = [2, 5]
multiply_by_two(list)
# list is [4, 10]!
```

EXAMPLE

```
def add_to_first(the_list, adder_list):
  for i in range(len(the_list)):
    the_list[i] = the_list[i] + adder_list[i]


the_list = [3, 4, 5]
add_to_first_list(the_list, [1, 2, 3])
# the_list is now [4, 6, 8]
```

OPERATORS

```
[1, 2] + [3, 4]
```
→ `[1, 2, 3, 4]`

```
[1, 2] * 3
```
→ `[1, 2, 1, 2, 1, 2]`

```
[1, 2] == [1, 2]
```
→ `True`

```
[1, 2] < [1, 3]
```
→ `True`

```
[1, 2] is [1, 2]
```
→ `False`

```
a = [1, 2]
yes = a is a
```

Pairwise comparison.

USING RANGE TO CREATE LISTS

```
range(5)
```
→ `0, 1, 2, 3, 4`

```
list(range(5))
```
→ `[0, 1, 2, 3, 4]`

```
list(range(5, 25, 5))
```
→ `[5, 10, 15, 20]`

Range can only produce linear sequences.

```
????????
```
→ `[1, 4, 9, 16]`

List comprehension to the rescue!

LIST COMPREHENSION

Expression creating a list based on a sequence.

```
[ <expr> for variable in <seq-expr> ]
```

```
[i*i for i in range(1, 5)]
```
→ `[1, 4, 9, 16]`

```
the_list = []
for i in range(1, 5):
  the_list.append(i*i)
```

EXAMPLE

```
[i for i in range(50, 53)]
```
→ `[50, 51, 52]`

```
list(range(50, 53))
```
→ `[50, 51, 52]`

```
[50+i for i in range(3)]
```
→ `[50, 51, 52]`

EXAMPLE

```
words = "one two three four five six".split(" ")
```

```
[len(word) for word in words]
```

→ `[3, 3, 5, 4, 4, 3]`

LIST COMPREHENSION

Expression creating a list based on a sequence.

```
[ <expr> for variable in <seq-expr> if <expr> ]
```

```
[i*i for i in range(1, 5) if i % 2 == 0]
```
→ `[4, 16]`

```
the_list = []
for i range(1, 5):
  if i % 2 == 0:
    the_list.append(i*i)
```

EXAMPLE

```
words = "one two three four five six".split(" ")
```

```
["_"+word+"_" for word in words if len(word) == 3]
```

→ ["_one_", "_two_", "_six_"]