Python3 Lists: Everything you need to know Cheat Sheet

by Nima (nimakarimian) via cheatography.com/113429/cs/23511/

Summing and multiplying

```
nums = [1, 2, 3]
print(nums + [4, 5, 6])
print(nums * 3)
```

Lists can be added and multiplied in the same way as strings.

"insert" FUNCTION

```
words = ["Python", "fun"]
index = 1
words.insert(index, " is")
print( words)
------ --- -------
>>>
['Python', 'is', 'fun']
>>>
```

insert method is similar to append, except that it allows you to insert a new item at any position in the list, as opposed to just at the end.

"range" FUNCTION

```
numbers = list(range(5, 20, 2))
print( num bers)
------ --- --- --- --- -------
>>>
[5, 7, 9, 11, 13, 15, 17, 19]
>>>
```

*** The range function creates a sequential list of numbers.**
*If range is called with one argument, it produces an object with values from 0 to that argument.
If it is called with two arguments, it produces values from the first to the second.
*range can have a third argument, which determines the **interval** of the sequence produced

ALL & ANY

```
nums = [55, 44, 33, 22, 11]
if all([i > 5 for i in nums]):
      pri nt( "All larger than
5")
if any([i % 2 == 0 for i in
nums]):
      pri nt( "At least one is
even")
```

Often used in conditional statements, all and any take a list as an argument, and return True if all or any (respectively) of their arguments evaluate to True (and False otherwise).

IN and NOT operator

```
words = ["spam", "egg", "spam",
"sausage"]
print( " spa m" in words)
#RETURNS TRUE
------ --- --- --- --- --- --
- --- --- --- --- ------
nums = [1, 2, 3]
print(not 4 in nums) #RETURNS
TRUE
print(4 not in nums)
```

The in operator is also used to determine whether or not a string is a substring of another string.

"index" FUNCTION

```
letters = ['p', 'q', 'r', 's',
'p', 'u']
print( let ter s.i nde x('r'))
print( let ter s.i nde x('z'))
------ --- --- --- --- --- --
- --- -------
>>>
```

"index" FUNCTION (cont)

> 2
ValueError: 'z' is not in list
>>>

index method finds the first occurrence of a list item and returns its index.

List comprehensions

```
cubes = [i**3 for i in range(5)]
print( cubes)
>>>
[0, 1, 8, 27, 64]
>>>
A list compre hension can also
contain an if statement to
enforce a condition on values in
the list.
evens=[i2 for i in range(10) if
i2 % 2 == 0]
print( evens)
>>>
[0, 4, 16, 36, 64]
>>>
```

Trying to create a list in a very extensive range will result in a MemoryError.

ENUMERATE

```
nums = [55, 44, 33, 22, 11]
for v in enumer ate (nums):
      pri nt(v)
------ --- --- --- --- -------
(0, 55)
(1, 44)
(2, 33)
(3, 22)
(4, 11)
```

The function enumerate can be used to iterate through the values and indices of a list simultaneously.

"append" FUNCTION

```
nums = [1, 2, 3]
nums.append(4)
print( nums)
------ --- --- --- --- ------
>>>
[1, 2, 3, 4]
>>>
```

This adds an item to the end of an existing list.

"Len" FUNCTION

```
nums = [1, 3, 5, 2, 4]
print( len (nums))
------ --- --- --- --- --- --
- -------
>>>
5
>>>
```

List slicing 1

```
squares = [0, 1, 4, 9, 16, 25,
36, 49, 64, 81]
print( squ are s[2:6])
print( squ are s[3:8])
------ ------
[4, 9, 16, 25]
[9, 16, 25, 36, 49]
```

Basic list slicing involves indexing a list with two colon-separated integers.

List slicing 2

```
squares = [0, 1, 4, 9, 16, 25,
36, 49, 64, 81]
print( squ are s[::2])
print( squ are s[2 :8:3]).
...................
>>>
[0, 4, 16, 36, 64]
[4, 25]
>>>
```

List slicing 2 (cont)

> ....

Negative values can be used in list slicing (and normal list indexing). When negative values are used for the first and second values in a slice (or a normal index), they count from the end of the list.

squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[1:-1])
>>>
[1, 4, 9, 16, 25, 36, 49, 64]
>>>

If a negative value is used for the step, the slice is done backwards.
Using [::-1] as a slice is a common and idiomatic way to reverse a list.

List slices can also have a third number, representing the step, to include only alternate values in the slice.