Python Bootcamp & Masterclass

modifying lists

gknxt

modifying existing element(s)

The index notation can be used to modify element(s) of a list as well as to extract element(s) from a list.

```
x = [1, 2, 3, 4]
x[2] = 'c'
x
```

```
[1, 2, 'c', 4]
```

```
y = ['a', 'b', 'c', 'd']
y[1] = [1, 2, 3]
y
```

```
['a', [1, 2, 3], 'c', 'd']
```

```
z = ['a', 'b', 'c', 'd']
z[1: ] = 'x'
z
```

```
['a', 'x']
```

```
x = [1, 2, 3, 4]
x[2:] = [8]
x
```

```
[1, 2, 8]
```

appending an element

The `list.append(x)` method appends the element `x` to the end of `list`.

It modifies list in place and returns `None`

```
a = []
a.append(42)
a
a.append('Hello')
a
```

```
[42, 'Hello']
```

```
b = [1, 2, 3]
b.append(b)     # appending a list to itself creates a circular reference: [...]
b
b[3][3][3][3][3][3][3][3][3][3]
```

```
[1, 2, 3, [...]]
```

```
b = [1, 2, 3]
b.append(b[:])     # appending a list to itself without a circular reference
b
len(b)
b[3][2]
```

```
[1, 2, 3, [1, 2, 3]]

4

3
```

inserting an element

The `list.insert(i, x)` method inserts the element `x` at position `i`. It modifies list in place and returns `None`

```
k = [1, 2, 3]
k.insert(0, 'a')
k
```

```
['a', 1, 2, 3]
```

```
n = [1, 2, 3]
n.insert(3, 'a')
n
```

```
[1, 2, 3, 'a']
```

```
l = [1, 2, 3]
l.insert(-1, 'a')
l
```

```
[1, 2, 'a', 3]
```

```
p = [1, 2, 3]
p.insert(100, 'a')
p
```

```
[1, 2, 3, 'a']
```

concatinating lists

To concatenate two strings, the most efficient method is to use `list.extend(iterable)` To concatenate multiple strings, the preferred way is to use unpacking operator (`*`) Lists can also be concatenated by using concatenation operator (`+`) or `Itertools.chain()` method.

```
a = [1, 2, 3]
b = [4, 5, 6]
a.extend(b)
a
```

```
[1, 2, 3, 4, 5, 6]
```

```
c = [1, 2, 3]
d = [4, 5, 6]
c = [*c, *d]
c
```

```
[1, 2, 3, 4, 5, 6]
```

```
e = [1, 2, 3]
f = [4, 5, 6]
e = e + f
e
```

```
[1, 2, 3, 4, 5, 6]
```

```
import itertools

g = [1, 2, 3]
h = [4, 5, 6]
h = list(itertools.chain(g, h))
h
```

```
[1, 2, 3, 4, 5, 6]
```

reverse( )

The `list.reverse()` method reverses the order of items of the `list` in place and returns `None`

```
g = [1, 2, 3]
g.reverse()
g
```

```
[3, 2, 1]
```

```
h = [1, 2, 3]
print(f'list h before reverse      = {h} and its id is: {hex(id(h))}')
h.reverse()
print(f'list h after reverse       = {h} and its id is: {hex(id(h))}')
h.reverse()
print(f'list h after reverse twice = {h} and its id is: {hex(id(h))}')
```

```
list h before reverse      = [1, 2, 3] and its id is: 0x1bc73bcba40
list h after reverse       = [3, 2, 1] and its id is: 0x1bc73bcba40
list h after reverse twice = [1, 2, 3] and its id is: 0x1bc73bcba40
```

```
k = [1, 2, 3]
print(f'list k = {k} and its id is: {hex(id(k))}')
l = k[ : : -1]
print(f'list l = {l} and its id is: {hex(id(l))}')
m = k[ : : -1][ : : -1]
print(f'list m = {m} and its id is: {hex(id(m))}')
```

```
list k = [1, 2, 3] and its id is: 0x1bc73c8ec40
list l = [3, 2, 1] and its id is: 0x1bc73cac9c0
list m = [1, 2, 3] and its id is: 0x1bc73c8e980
```

reversed( )

The `reversed(seq)` function returns reverse iterator and the `list()` method can be used to get the list in reverse order. The original `seq` will not be changed.

```
p = [1, 2, 3]
print(f'list p before reversed( ) = {p} and its id is: {hex(id(p))}')
q = list(reversed(p))
print(f'list p after reversed( )  = {p} and its id is: {hex(id(p))}')
print(f'list q after reversed( )  = {q} and its id is: {hex(id(q))}')
```

```
list p before reversed( ) = [1, 2, 3] and its id is: 0x1bc73ca8a00
list p after reversed( )  = [1, 2, 3] and its id is: 0x1bc73ca8a00
list q after reversed( )  = [3, 2, 1] and its id is: 0x1bc73ca9cc0
```

Online Resources

For best python resources, please visit:

gknxt.com/python/

gknxt

Python
Bootcamp
& Masterclass

Thank You

for your Rating & Review

gknxt