Every Boilermaker Engineer Codes: 101 Entry-Level Programming in Python

Lecture 07

Dr. John H. Cole
<jhcole@purdue.edu>

PURDUE UNIVERSITY®

COLLEGE OF ENGINEERING

Spring 2021

Part I

List and Tuple Sequences

Sequences

- can contain multiple items
- items have a clearly defined order
- basic types: `range`, `list`, and `tuple`
- additional types: `str` ings, `bytes`, and `bytearray`

Terminal

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

Lists vs. Tuples: Syntax

`list`

- enclosed in brackets `[]`
- items separated by commas

Terminal

```
>>> l = [1, 2, 3]
>>> l
[1, 2, 3]
>>> type(l)
<class 'list'>
```

`tuple`

- enclosed in parentheses `()`
- items separated by commas

Terminal

```
>>> t = (1, 2, 3)
>>> t
(1, 2, 3)
>>> type(t)
<class 'tuple'>
```

Lists vs. Tuples: Syntax Examples

- empty parentheses are an empty tuple
- non-empty parentheses without a comma is not a tuple (it is an expression)

Terminal

```
>>> type([])
<class 'list'>
>>> type([1])
<class 'list'>
>>> type([1+2]*3)
<class 'list'>
>>> [1+2]*3 # repetition
[3, 3, 3]
```

Terminal

```
>>> type(())
<class 'tuple'>
>>> type((1))
<class 'int'>
>>> type((1+2)*3)
<class 'int'>
>>> type((1,))
<class 'tuple'>
```

Heterogeneous Sequences (Lists and Tuples)

- Lists and tuples can have items of different types

Terminal

```
>>> l = ['a', (1, True)]
>>> l
['a', (1, True)]
```

Terminal

```
>>> t = ('a', [1, True])
>>> t
('a', [1, True])
```

Lists vs. Tuples: Conversions

`list(x)` returns a list

Terminal

```
>>> list((1, 2, 3))
[1, 2, 3]
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list('spam')
['s', 'p', 'a', 'm']
```

`tuple(x)` returns a tuple

Terminal

```
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> tuple(range(5))
(0, 1, 2, 3, 4)
>>> tuple('spam')
('s', 'p', 'a', 'm')
```

Lists vs. Tuples: Mutability

Lists are mutable

- i.e. can be changed

Terminal

```
>>> l = [1, 2, 3]
>>> l
[1, 2, 3]
>>> l[0] = 5
>>> l
[5, 2, 3]
>>> l[1] = 4
>>> l
[5, 4, 3]
>>>
```

Tuples are immutable

- i.e. cannot be changed

Terminal

```
>>> t = (1, 2, 3)
>>> t
(1, 2, 3)
>>> t[0] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not
support item assignment
>>> t
(1, 2, 3)
```

Mutable Sequences (list and bytearray)

Mutable sequences (`list` and `bytearray`) can do these things that immutable sequences (`tuple`, `bytes`, `str`ing, `range`) cannot do.

`s[i] = y`	replace item i with x
`s.copy()`	make a shallow copy
`s.append(x)`	add and item to the end
`s.extend(t)`	add a sequence of items
`s.insert(i, x)`	insert item x at i
`del s[i]`	delete an item by index
`s.clear()`	remove all items
`s.remove(x)`	remove first item equal to x
`s.pop([i])`	retrieve and remove item i
`s.reverse()`	reverse the order of s
`s.sort()`	sort for homogeneous lists only

For mutable sequence `s`, index `i`, iterable object `t` and arbitrary object `x`.

Basic Indexing

Index a number specifying the position of an item in a sequence

`s[i]` return the item at index `i`

- all sequences can be indexed
- enables individual access to sequence items
- first item is at index 0
- last item is at index (length - 1)
- indices >= sequence length are invalid (IndexError)

Terminal

```
>>> l = ['a', 'b', 'c']
>>> l[0]
'a'
>>> l[2]
'c'
```

Terminal

```
>>> s = 'spam'
>>> s[0]
's'
>>> s[3]
'm'
```

Negative Indexing

negative indices count from the end of the sequence

- last item is at `-1`
- first item is at `-length`
- negative indices > sequence length are invalid (IndexError)

Terminal

```
>>> l = ['a', 'b', 'c']
>>> l[-1]
'c'
>>> l[-3]
'a'
```

Terminal

```
>>> s = 'spam'
>>> s[-1]
'm'
>>> s[-4]
's'
```

Negative Indexing Exercise

Complete this program by adding a for loop to print the values of the list in reverse order using negative indices.

Terminal

```
$ python neg_index.py
100
35
93
62
```

Editor - neg_index.py

```
1 v = [ 62, 93, 35, 100]
2
3
4
5
6
7
8 # End
```

Negative Indexing Exercise

Complete this program by adding a for loop to print the values of the list in reverse order using negative indices.

Terminal

```
$ python neg_index.py
100
35
93
62
```

Editor - neg_index.py

```
1 v = [ 62, 93, 35, 100]
2
3 for r in range(1,5):
4   print(v[-r])
5
6 for r in range(-1,-5,-1):
7   print(v[r])
8 # End
```

Multidimensional Indexing

multidimensional indexing is used to reference elements in nested sequences

- use one index per dimension
- first index commonly called rows
- second index commonly called columns

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} \texttt{l[0][0]} & \texttt{l[0][1]} & \texttt{l[0][2]} \\ \texttt{l[1][0]} & \texttt{l[1][1]} & \texttt{l[1][2]} \end{bmatrix}$$

Terminal

```
>>> l = [[1,2,3],[4,5,6]]
>>> l[0]
[1, 2, 3]
>>> l[1]
[4, 5, 6]
```

Terminal

```
>>> l[0][0]
1
>>> l[1][2]
6
>>>
```

Multidimensional Indexing Exercise

Complete this program to print the values of the two dimensional list v as a 4×3 matrix as shown below. Hint: use nested for loops.

Terminal

```
$ python 2D_rand.py
  62   93   35
 100   62   16
  63   76   76
  16    1   69
```

Editor - 2D_rand.py

```
1  v = [[ 62, 93, 35],
2       [100, 62, 16],
3       [ 63, 76, 76],
4       [ 16,  1, 69]]
5
6
7
8
9
10
11
12 # End
```

Multidimensional Indexing Exercise

Complete this program to print the values of the two dimensional list v as a 4×3 matrix as shown below. Hint: use nested for loops.

Terminal

```
$ python 2D_rand.py
  62  93  35
 100  62  16
  63  76  76
  16   1  69
```

Editor - 2D_rand.py

```
1  v = [[ 62, 93, 35],
2       [100, 62, 16],
3       [ 63, 76, 76],
4       [ 16,  1, 69]]
5
6  rows, cols = 4,3
7  for r in range(rows):
8    for c in range(cols):
9      n = v[r][c]
10     print(f'{n:4d}', end='')
11   print()
12 # End
```

Slicing

Slice selecting a subset of items in a sequence

`s[i]`	`i`th item of `s`
`s[i:j]`	slice from `i` to `j`
`s[i:j:k]`	slice from `i` to `j` with step `k`

(for some sequence `s`, and integers `i`, `j`, and `k`)

`i`:

- default: 0 for positive step
- default: -1 for negative step

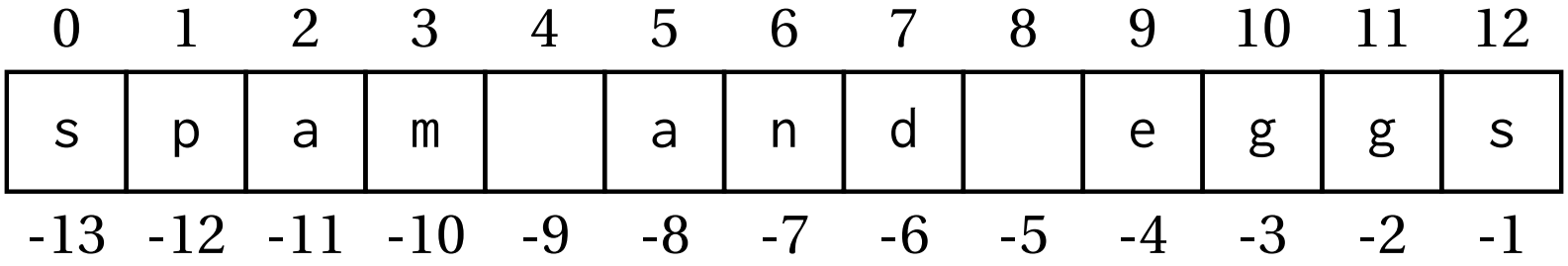
`j`:

- always excluded from the slice
- default: length for positive step
- default: -(length+1) for negative step

`k`:

- default: 1

Slicing Examples



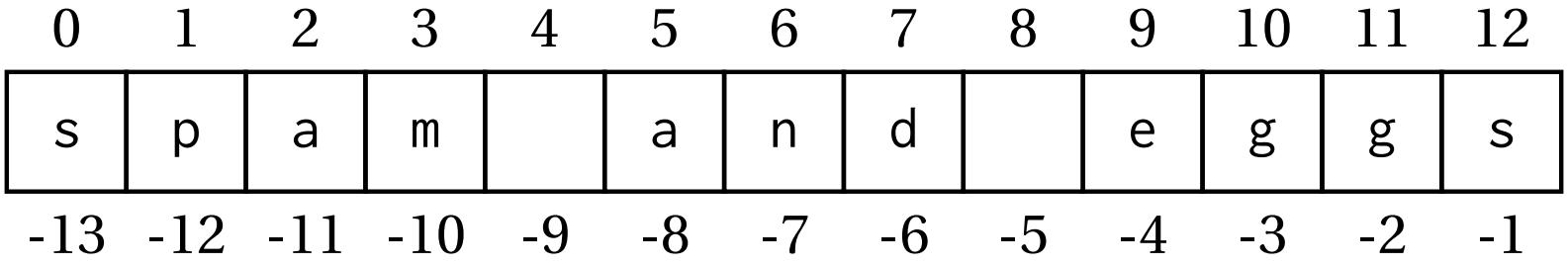
Terminal

```
>>> s = 'spam and eggs'
>>> s[3]
'm'
>>> s[5:8]
'and'
>>> s[3:11:2]
'made'
>>>
```

Terminal

```
>>> s[5:]
'and eggs'
>>> s[:8]
'spam and'
>>> s[::5]
'sag'
>>> s[7::-1]
'dna maps'
```

Slicing More Examples



Terminal

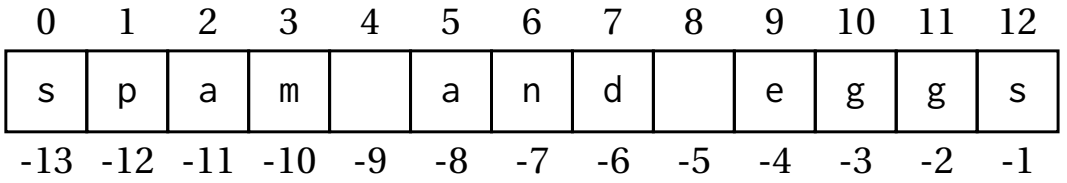
```
>>> s = 'spam and eggs'
>>> s[-4]
'e'
>>> s[-8:-5]
'and'
>>> s[-10:-2:2]
'made'
>>>
```

Terminal

```
>>> s[-8:]
'and eggs'
>>> s[:-5]
'spam and'
>>> s[-3::-5]
'gas'
>>> s[-6:-14:-1]
'dna maps'
```

Slicing Exercise

Complete this program by modifying the slices to make each comparison evaluate to `True`.



Terminal

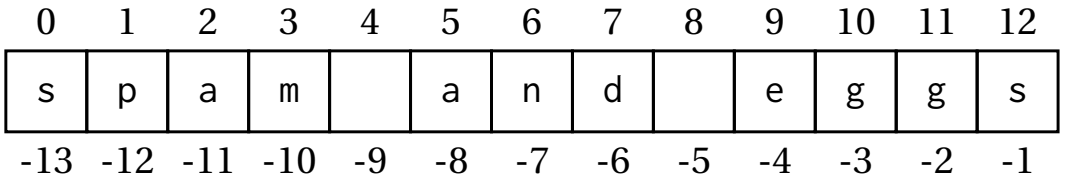
```
$ python slices.py
1: False
2: False
3: False
4: False
5: False
6: False
```

Editor - slices.py

```
1 s = 'spam and eggs'
2
3 print('1:', s[:] == 'm')
4 print('2:', s[:] == 'pam')
5 print('3:', s[:] == 'mad')
6 print('4:', s[:] == 'map')
7 print('5:', s[:] == 'ged')
8 print('6:', s[:] == 'damp')
```

Slicing Exercise

Complete this program by modifying the slices to make each comparison evaluate to `True`.



Terminal

```
$ python slices.py
1: True
2: True
3: True
4: True
5: True
6: True
```

Editor - slices.py

```
1 s = 'spam and eggs'
2
3 print('1:', s[3] == 'm')
4 print('2:', s[1:4] == 'pam')
5 print('3:', s[3:8:2] == 'mad')
6 print('4:', s[3:0:-1] == 'map')
7 print('5:', s[-2:6:-2] == 'ged')
8 print('6:', s[7::-2] == 'damp')
```

Concatenation

Concatenate to join two or more things together

`s1 + s2` make a new sequence by concatenating `s1` and `s2`

- sequences must be of the same type
- does *not* work for `range`

Terminal

```
>>> [1,2,3]+[4,5,6]
[1, 2, 3, 4, 5, 6]
>>> (1,2,3)+(4,5,6)
(1, 2, 3, 4, 5, 6)
>>> '1,2,3' + '4,5,6'
'1,2,34,5,6'
```

Terminal

```
>>> [1,2,3] + (4,5,6)
Traceback ...
TypeError: ...
>>> range(3) + range(3)
Traceback ...
TypeError: ...
```

Repetition

`s*n` make a new sequence by concatenating `n` copies of `s`

- can also use `n*s`
- non-positive `n` results in an empty sequence
- does *not* work for `range`

Terminal

```
>>> [1,2,3]*2
[1, 2, 3, 1, 2, 3]
>>> (1,2)*3
(1, 2, 1, 2, 1, 2)
>>> '1,2,3'*3
'1,2,31,2,31,2,3'
```

Terminal

```
>>> 2*[1,2,3]
[1, 2, 3, 1, 2, 3]
>>> (1,2,3)*0
()
>>> 'abc'*-2
''
```

s += t

`s += t` extend s with the contents of `t`

- same as `s.extend(t)` if s is mutable
- creates a new sequence if s is immutable

Terminal

```
>>> l = [1]
>>> l += [2,3]
>>> l
[1, 2, 3]
>>> l += 'abc'
>>> l
[1, 2, 3, 'a', 'b', 'c']
```

Terminal

```
>>> l = ['a']
>>> l += ('b', 'c')
>>> l
['a', 'b', 'c']
>>> l += (1,)
>>> l
['a', 'b', 'c', 1]
```

s *= n

`s *= n` updates s with its contents repeated n times

- non-positive n clears the sequence
- changes s in place if it is mutable
- creates a new sequence if s is immutable

Terminal

```
>>> l = [1,2,3]
>>> l *= 2
[1, 2, 3, 1, 2, 3]
>>> l = '1,2,3'
>>> l *= 3
'1,2,31,2,31,2,3'
```

Terminal

```
>>> l = [1,2,3]
>>> l *= 0
[]
>>> l = '1,2,3'
>>> l *= -2
''
```

Inclusion

`x in s` `True` if an item in `s` is equal to `x`, `False` otherwise

`x not in s` `False` if an item in `s` is equal to `x`, `True` otherwise

- for arbitrary `x` and sequence `s`
- when `s` is a `str`ing sequence, `x` can be a substring

Terminal

```
>>> 3 in [1,2,3]
True
>>> 4 in range(4)
False
>>> 'a' in 'spam'
True
>>> 'am' in 'spam'
True
```

Terminal

```
>>> 'd' not in ['a', 'b', 'c']
True
>>> (1,2) not in ['a', (1,2)]
False
>>> 'S' not in 'spam'
True
>>> 'eggs' not in 'spam'
True
```

`len(s)` AND `s.count(x)`

`len(s)` return the length of `s`

`s.count(x)` return the total number of occurrences of `x` in `s`

- for sequence `s` and arbitrary `x`

Terminal

```
>>> len([])
0
>>> len(range(4))
4
>>> len('spam')
4
```

Terminal

```
>>> [[], [], []].count([])
3
>>> range(100).count(42)
1
>>> 'eggs'.count('g')
2
```

min(s) AND max(s)

`min(s)` return the smallest item of `s`

`max(s)` return the largest item of `s`

- for sequence `s`
- items in sequence must have valid comparison operators

Terminal

```
>>> min('spam')
'a'
>>> min(range(5))
0
>>> min([1, 'a', ()]
Traceback ...
TypeError: ...
```

Terminal

```
>>> max('spam')
's'
>>> max(range(5))
4
>>> max([1, 'a', ()]
Traceback ...
TypeError: ...
```

sum(s[, n])

`sum(s)` add the items in `s` together

`sum(s, n)` add the items in `s` to `n`

- usually for numeric sequence `s`, and number `n`
- `n` defaults to 0, cannot be a `str`ing

Terminal

```
>>> sum(range(1000))
499500
>>> sum([9, 29, 4])
42
>>> from math import *
>>> sum([pi, e, tau])
12.143059789228424
```

Terminal

```
>>> sum(range(1000),499)
499999
>>> sum([9, 29, 4],-10)
32
>>> s = [['a'],['m']]
>>> sum(s, ['s','p'])
['s', 'p', 'a', 'm']
```

s.index(x[, i[, j]])

`s.index(x)`	return index of first occurrence of `x` in `s`
`s.index(x, i)`	search at or after index `i`
`s.index(x, i, j)`	search at or after index `i` and before index `j`

- raises an error if `x` is not found

Terminal

```
>>> s = (3, 'a', [2, 5])
>>> s.index('a')
1
>>> s.index([2,5])
2
>>> s.index(2)
Traceback ...
ValueError: ...
```

Terminal

```
>>> s = 'spam and eggs'
>>> s.index('a')
2
>>> s.index('a', 3, 8)
5
>>> s[3:8].index('a')
2
>>>
```

sorted(s[, key=None][, reverse=False])

`sorted(s)`	return a sorted list from the items in `s`
`sorted(s, key=f)`	compare items using function `f`
`sorted(s, reverse=True)`	reverse order

- use ascending order by default
- does not change the sequence `s`
- items in sequence must have valid comparison operators

Terminal

```
>>> sorted('spam')
['a', 'm', 'p', 's']
>>> sorted((3,1,-2))
[-2, 1, 3]
```

Terminal

```
>>> sorted('spam', reverse=True)
['s', 'p', 'm', 'a']
>>> sorted((3,1,-2), key=abs)
[1, -2, 3]
```

reversed(s)

`reversed(s)` return a reverse iterator over s

- does not change the sequence s

Terminal

```
>>> reversed([1,2,3])
<list_reverseiterator object at 0x7f52bf899d68>
>>> list(reversed([1,2,3]))
[3, 2, 1]
>>> r = reversed([1,2,3])
>>> next(r)
3
>>> next(r)
2
>>> next(r)
1
```

Processing Exercise 1

Complete this program by calculating the min, max, sum and mean of the list of numbers.

Terminal

```
$ python stats.py
min: 13
max: 42
sum: 225
mean: 28.12
```

Editor - stats.py

```
1 l = [31, 27, 13, 42, 31, 27, 31, 23]
2
3 print('min:', )
4 print('max:', )
5 print('sum:', )
6 print(f'mean: {:.2f}')
```

Processing Exercise 1

Complete this program by calculating the min, max, sum and mean of the list of numbers.

Terminal

```
$ python stats.py
min: 13
max: 42
sum: 225
mean: 28.12
```

Editor - stats.py

```
1 l = [31, 27, 13, 42, 31, 27, 31, 23]
2
3 print('min:', min(l))
4 print('max:', max(l))
5 print('sum:', sum(l))
6 print(f'mean: {sum(l)/len(l):.2f}')
```

Processing Exercise 2

Complete this program by calculating the median of the list of numbers. The median is the middle value of the list. If there are an even number of values, the median is the average of the two values closest to the middle.

Terminal

```
$ python stats2.py
median: 29.00
```

Editor - stats2.py

```
1  l = [31, 27, 13, 42, 31, 27, 31, 23]
2
3
4
5
6
7
8
9
10
11 print(f'median: {median:.2f}')
```

Processing Exercise 2

Complete this program by calculating the median of the list of numbers. The median is the middle value of the list. If there are an even number of values, the median is the average of the two values closest to the middle.

Terminal

```
$ python stats2.py
median: 29.00
```

Editor - stats2.py

```
1  l = [31, 27, 13, 42, 31, 27, 31, 23]
2
3  l = sorted(l)
4
5  n = len(l)
6  if n % 2: # odd number of numbers
7      median = l[(n-1)//2]
8  else: # even number of numbers
9      median = (l[n//2-1] + l[n//2])/2
10
11 print(f'median: {median:.2f}')
```

l.append(x)

`l.append(x)` appends `x` to the end of the list `l`

- only adds a single item

Terminal

```
>>> l = [1]
>>> l.append(2)
>>> l
[1, 2]
>>> l.append(True)
>>> l
[1, 2, True]
```

Terminal

```
>>> l = ['a']
>>> l.append('bcd')
>>> l
['a', 'bcd']
>>> l.append(['e', 'f'])
>>> l
['a', 'bcd', ['e', 'f']]
```

l.extend(t)

`l.extend(t)` extend `l` with the contents of `t`

- `t` must be iterable
- adds as many items as are in `t`
- same as `s += t` for mutable sequences (e.g. `list`)

Terminal

```
>>> l = [1]
>>> l.extend([2,3])
>>> l
[1, 2, 3]
>>> l.extend('abc')
>>> l
[1, 2, 3, 'a', 'b', 'c']
```

Terminal

```
>>> l = ['a']
>>> l.extend(('b', 'c'))
>>> l
['a', 'b', 'c']
>>> l.extend((1,))
>>> l
['a', 'b', 'c', 1]
```

l.insert(i, x)

`l.insert(i, x)` insert `x` into `l` at index `i`

- shifts existing items toward the end of the list
- if `i >= len(l)` insert at the end of the list
- if `i < -len(l)` insert at the beginning of the list

Terminal

```
>>> l = ['t']
>>> l.insert(0, 'o')
>>> l
['o', 't']
>>> l.insert(1, 'n')
>>> l
['o', 'n', 't']
```

Terminal

```
>>> l.insert(100, 'y')
>>> l
['o', 'n', 't', 'y']
>>> l.insert(-100, 'm')
>>> l
['m', 'o', 'n', 't', 'y']
>>>
```

del l[[i][:[j][:[k]]]]

`del l`	deletes the entire list; `l` becomes undefined
`del l[i]`	removes item at index `i` from the list
`del l[i:j]`	removes items in slice `[i:j]` from the list
`del l[i:j:k]`	removes items in slice `[i:j:k]` from the list

Terminal

```
>>> l = [10, 11, 12, 13, 14]
>>> del l[2]
>>> l
[10, 11, 13, 14]
>>> l = [10, 11, 12, 13, 14]
>>> del l[-1]
>>> l
[10, 11, 12, 13]
```

Terminal

```
>>> l = [10, 11, 12, 13, 14]
>>> del l[1:3]
>>> l
[10, 13, 14]
>>> l = [10, 11, 12, 13, 14]
>>> del l[::2]
>>> l
[11, 13]
```

l.clear()

`l.clear()` removes all the items from `l`

- does not delete `l`
- same as del `s[:]`

Terminal

```
>>> l = [10, 11, 12, 13, 14]
>>> l.clear()
>>> l
[]
>>> l = [10, 11, 12, 13, 14]
>>> del l[:]
>>> l
[]
```

l.remove(x)

`l.remove(x)` removes the first item in `l` where `l[i]` is equal to `x`

- raises an error when `x` is not found

Terminal

```
>>> l = [10, 11]*3
>>> l.remove(11)
>>> l
[10, 10, 11, 10, 11]
>>> l.remove(12)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  ValueError: list.remove(x): x not in list
```

l.pop([i])

`l.pop()` return the item at the end of `l` and remove it from `l`

`l.pop(i)` return the item at index `i` and remove it from `l`

Terminal

```
>>> l = list(range(10,15))
>>> l.pop()
14
>>> l
[10, 11, 12, 13]
>>> l.pop(2)
12
>>> l
[10, 11, 13]
```

Assigning to a Slice

`l[i] = x` replace item `i` with `x`

`l[i:j] = x` replace slice `i:j` with contents of `x`

`l[i:j:k] = x` replace elements of slice `i:j:k` with those of `x`

Terminal

```
>>> l = [1, 2, 3, 4, 5]
>>> l[2] = ('a', 'b')
>>> l
[1, 2, ('a', 'b'), 4, 5]
>>> l = [1, 2, 3, 4, 5]
>>> l[1:4] = ('a', 'b')
>>> l
[1, 'a', 'b', 5]
```

Terminal

```
>>> l = [1, 2, 3, 4, 5]
>>> l[1::2] = ('a', 'b')
>>> l
[1, 'a', 3, 'b', 5]
>>> l = [1, 2, 3, 4, 5]
>>> l[::2] = ('a','b','c')
>>> l
['a', 2, 'b', 4, 'c']
```

l.copy()

`l.copy()` create a *shallow* copy of `l`

Terminal

```
>>> l1 = [1, 2, [3, 4]]
>>> l2 = l1
>>> l1[1] = 'a'
>>> l1
[1, 'a', [3, 4]]
>>> l2
[1, 'a', [3, 4]]
>>> l1[2][1] = 'b'
>>> l1
[1, 'a', [3, 'b']]
>>> l2
[1, 'a', [3, 'b']]
```

Terminal

```
>>> l1 = [1, 2, [3, 4]]
>>> l2 = l1.copy()
>>> l1[1] = 'a'
>>> l1
[1, 'a', [3, 4]]
>>> l2
[1, 2, [3, 4]]
>>> l1[2][1] = 'b'
>>> l1
[1, 'a', [3, 'b']]
>>> l2
[1, 2, [3, 'b']]
```

l.sort()

`l.sort()` sort the items of `l` in place

- items are organized from smallest to largest
- returns NoneType

Terminal

```
>>> l = [2, 8, 0, 6, 2]
>>> l.sort()
>>> l
[0, 2, 2, 6, 8]
>>> l = ['s', 'p', 'a', 'm']
>>> l.sort()
>>> l
['a', 'm', 'p', 's']
```

l.reverse()

`l.reverse()` reverses the items of `l` in place

- returns NoneType

Terminal

```
>>> l = [1, 2, [3, 4]]
>>> l.reverse()
>>> l
[[3, 4], 2, 1]
>>> l.reverse()
>>> l
[1, 2, [3, 4]]
```


Thanks for watching!