Lists Advanced

Lists Advanced

Copying Lists, List Methods, and 2D Lists

List Review

List Review

```
>>> vowels = ["a", "e", "i", "o", "u"]
>>> print(vowels)
["a", "e", "i", "o", "u"]
```

List Review

```
>>> vowels = ["a", "e", "i", "o", "u"]
>>> print(vowels)
["a", "e", "i", "o", "u"]

>>> myList = [0] * 5
>>> print(myList)
[0, 0, 0, 0, 0]
```

List Review

```
>>> vowels = ["a", "e", "i", "o", "u"]
>>> print(vowels)
["a", "e", "i", "o", "u"]

>>> myList = [0] * 5
>>> print(myList)
[0, 0, 0, 0, 0]

>>> vowels = ["a", "e", "i", "o", "u"]
>>> vowels = vowels + ["y"]
>>> print(vowels)
["a", "e", "i", "o", "u", "y"]
```

Basic list indexing

To access an individual item in the list, use its *index*:

```
>>> vowels = ["a", "e", "i", "o", "u"]
                0    1    2    3    4

>>> print(vowels[0])
a

>>> print(vowels[4])
u

>>> print(vowels[5])
IndexError:  list index out of range
```

Lists and Loops

Lists and Loops

```
>>> for letter in vowels:
...     print(letter)

a
e
i
o
u
```

Lists and Loops

```
>>> for letter in vowels:
...     print(letter)

a
e
i
o
u

for i in range(len(vowels)):
    print(vowels[i])

a
e
i
o
u
```

Advanced indexing

Use negative numbers to index from the back of the list:

```
>>> vowels = ["a", "e", "i", "o", "u"]
```

$$-5 \quad -4 \quad -3 \quad -2 \quad -1$$

```
for i in range(1, len(vowels)+1):
    print(vowels[-i])
```

```
u
o
i
e
a
```

Slicing

Can create a sublist by "slicing" a list:

```
>>> vowels = ["a", "e", "i", "o", "u"]

>>> print(vowels[1:3])
[e, i]

>>> print(vowels[:3])
[a, e, i]

>>> print(vowels[3:])
[o, u]

>>> print(vowels[:])
[a, e, i, o, u]
```

List Functions

List Functions

```
>>> numbers = [1, 1, 2, 1]
```

List Functions

```
>>> numbers = [1, 1, 2, 1]
>>> len(numbers)
```

List Functions

```
>>> numbers = [1, 1, 2, 1]
>>> len(numbers)
4
```

List Functions

```
>>> numbers = [1, 1, 2, 1]
>>> len(numbers)
4

>>> numbers = [1, 1, 2, 1]
```

List Functions

```
>>> numbers = [1, 1, 2, 1]
>>> len(numbers)
4

>>> numbers = [1, 1, 2, 1]
>>> sum(numbers)
```

List Functions

```
>>> numbers = [1, 1, 2, 1]
>>> len(numbers)
4

>>> numbers = [1, 1, 2, 1]
>>> sum(numbers)
5
```

List Functions

```
>>> numbers = [1, 1, 2, 1]
>>> len(numbers)
4

>>> numbers = [1, 1, 2, 1]
>>> sum(numbers)
5

vowels = ["a", "e", "i", "o", "u"]
```

List Functions

```
>>> numbers = [1, 1, 2, 1]
>>> len(numbers)
4

>>> numbers = [1, 1, 2, 1]
>>> sum(numbers)
5

vowels = ["a", "e", "i", "o", "u"]
```

- `min(vowels)` → `a`
- `max(vowels)` → `y`

Copying Lists

```
>>> myList = [0, 1, 2, 3]
```

Copying Lists

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = myList
```

Copying Lists

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = myList
>>> listCopy[0] = 1
```

Copying Lists

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = myList
>>> listCopy[0] = 1
>>> print(myList)
```

Copying Lists

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = myList
>>> listCopy[0] = 1
>>> print(myList)
[1, 1, 2, 3]
```

✗

List Pointers

```
x = [0, 1, 2, 3]
y = x
y[0] = 1
```

List Pointers

```
x = [0, 1, 2, 3]
y = x
y[0] = 1
```

x $\rightarrow$

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x[0]	00000000
x[1]	00000001
⋮	⋮
11111111	00100000

List Pointers

```
x = [0, 1, 2, 3]
y = x
y[0] = 1
```

x , y $\rightarrow$

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x[0] , y[0]	00000000
x[1] , y[1]	00000001
⋮	⋮
11111111	00100000

List Pointers

```
x = [0, 1, 2, 3]
y = x
y[0] = 1
```

x , y $\rightarrow$

Memory:

Address	Contents
00000000	01101011
00000001	11001100
x[0] , y[0]	**00000001**
x[1] , y[1]	00000001
⋮	⋮
11111111	00100000

Copying Lists: Loop

Copying Lists: Loop

```
>>> myList = [0, 1, 2, 3]
```

Copying Lists: Loop

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [0] * len(myList)
```

Copying Lists: Loop

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [0] * len(myList)
>>> for i in range(len(myList)):
...     listCopy[i] = myList[i]
```

Copying Lists: Loop

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [0] * len(myList)
>>> for i in range(len(myList)):
...     listCopy[i] = myList[i]
>>> listCopy[0] = 1
```

Copying Lists: Loop

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [0] * len(myList)
>>> for i in range(len(myList)):
...     listCopy[i] = myList[i]
>>> listCopy[0] = 1
>>> print(myList)
```

Copying Lists: Loop

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [0] * len(myList)
>>> for i in range(len(myList)):
...     listCopy[i] = myList[i]
>>> listCopy[0] = 1
>>> print(myList)
[0, 1, 2, 3]
```

✓

Copying Lists: Comprenehsions

Copying Lists: Comprenehsions

```
>>> myList = [0, 1, 2, 3]
```

Copying Lists: Comprenehsions

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [item for item in myList]
```

Copying Lists: Comprenehsions

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [item for item in myList]
>>> listCopy[0] = 1
```

Copying Lists: Comprenehsions

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [item for item in myList]
>>> listCopy[0] = 1
>>> print(myList)
```

Copying Lists: Comprenehsions

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = [item for item in myList]
>>> listCopy[0] = 1
>>> print(myList)
[0, 1, 2, 3]
```

✓

Copying Lists: list.copy() method

Copying Lists: list.copy() method

```
>>> myList = [0, 1, 2, 3]
```

Copying Lists: list.copy() method

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = myList.copy()
```

Copying Lists: list.copy() method

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = myList.copy()
>>> listCopy[0] = 1
```

Copying Lists: list.copy() method

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = myList.copy()
>>> listCopy[0] = 1
>>> print(myList)
```

Copying Lists: list.copy() method

```
>>> myList = [0, 1, 2, 3]
>>> listCopy = myList.copy()
>>> listCopy[0] = 1
>>> print(myList)
[0, 1, 2, 3]
```

✓

Methods

Methods

A *method* is a type of function that is attached to a data structure.

Methods

A *method* is a type of function that is attached to a data structure.

Calling a method allows you to check or alter the contents of a data strucure (such as a list)

Methods

A *method* is a type of function that is attached to a data structure.

Calling a method allows you to check or alter the contents of a data strucure (such as a list)

Don't need to use "=" assignment with some methods

Methods

A *method* is a type of function that is attached to a data structure.

Calling a method allows you to check or alter the contents of a data strucure (such as a list)

Don't need to use "=" assignment with some methods

`myList.append(x)` ✓

Methods

A *method* is a type of function that is attached to a data structure.

Calling a method allows you to check or alter the contents of a data strucure (such as a list)

Don't need to use "=" assignment with some methods

`myList.append(x)` ✓

`myList = myList.append(x)` ✗

```
vowels = ["a", "e", "i", "o", "u"]
```

```
vowels = ["a", "e", "i", "o", "u"]
```

- ```
  vowels.append("y")
  ["a", "e", "i", "o", "u", "y"]
  ```
  ```

```
vowels = ["a", "e", "i", "o", "u"]
```

- ```
  vowels.append("y")
  ["a", "e", "i", "o", "u", "y"]
  ```
- ```
  vowels.insert("hello world", 0)
  ["hello world", "a", "e", "i", "o", "u", "y"]
  ```

```
vowels = ["a", "e", "i", "o", "u"]
```

- ```
  vowels.append("y")
  ["a", "e", "i", "o", "u", "y"]
  ```
- ```
  vowels.insert("hello world", 0)
  ["hello world", "a", "e", "i", "o", "u", "y"]
  ```
- ```
  vowels.remove("hello world")
  ["a", "e", "i", "o", "u", "y"]
  ```
  ```

```
vowels = ["a", "e", "i", "o", "u"]
```

- `vowels.append("y")`
  `["a", "e", "i", "o", "u", "y"]`
- `vowels.insert("hello world", 0)`
  `["hello world", "a", "e", "i", "o", "u", "y"]`
- `vowels.remove("hello world")`
  `["a", "e", "i", "o", "u", "y"]`
- `vowels.pop(-1)` (returns "u")
  `["a", "e", "i", "o", "u"]`

```
vowels = ["a", "e", "i", "o", "u"]
```

- `vowels.append("y")`
  `["a", "e", "i", "o", "u", "y"]`
- `vowels.insert("hello world", 0)`
  `["hello world", "a", "e", "i", "o", "u", "y"]`
- `vowels.remove("hello world")`
  `["a", "e", "i", "o", "u", "y"]`
- `vowels.pop(-1)` (returns "u")
  `["a", "e", "i", "o", "u"]`
- `vowels.index("i")`
  `2`

```
vowels = ["a", "e", "i", "o", "u"]
```

- `vowels.append("y")`
  `["a", "e", "i", "o", "u", "y"]`
- `vowels.insert("hello world", 0)`
  `["hello world", "a", "e", "i", "o", "u", "y"]`
- `vowels.remove("hello world")`
  `["a", "e", "i", "o", "u", "y"]`
- `vowels.pop(-1)` (returns "u")
  `["a", "e", "i", "o", "u"]`
- `vowels.index("i")`
  `2`
- `vowels.sort()`
  `["a", "e", "i", "o", "u"]`

```
vowels = ["a", "e", "i", "o", "u"]
```

- `vowels.append("y")`
  `["a", "e", "i", "o", "u", "y"]`
- `vowels.insert("hello world", 0)`
  `["hello world", "a", "e", "i", "o", "u", "y"]`
- `vowels.remove("hello world")`
  `["a", "e", "i", "o", "u", "y"]`
- `vowels.pop(-1)` (returns "u")
  `["a", "e", "i", "o", "u"]`
- `vowels.index("i")`
  `2`
- `vowels.sort()`
  `["a", "e", "i", "o", "u"]`

Read more:
https://docs.python.org/3/library/stdtypes.html#list

Dynamically expanding lists with append() method

Dynamically expanding lists with append() method

```
myList = []
```

Dynamically expanding lists with append() method

```
myList = []
userInput = input("Enter next item, or q to quit:  ")
```

Dynamically expanding lists with append() method

```
myList = []
userInput = input("Enter next item, or q to quit:  ")
while userInput != "q":
```

Dynamically expanding lists with append() method

```
myList = []
userInput = input("Enter next item, or q to quit:  ")
while userInput != "q":
    myList.append(userInput)
```

Dynamically expanding lists with append() method

```
myList = []
userInput = input("Enter next item, or q to quit:  ")
while userInput != "q":
    myList.append(userInput)
    userInput = input("Next item, q to quit:  ")
```

Concept Check!

Which of the following commands will alter the value of `myList`?
`myList = [1.0, 1.5, 2.0, 2.5, 3.0]`

1. `myList + [3.5]`
2. `myList.append(3.5)`
3. `myCopy = myList`
   `myCopy.append(3.5)`
4. `myCopy = myList.copy()`
   `myCopy.append(3.5)`

Concept Check!

Which of the following commands will alter the value of `myList`?

```
myList = [1.0, 1.5, 2.0, 2.5, 3.0]
```

1. `myList + [3.5]` ✗
2. `myList.append(3.5)`
3.
```
myCopy = myList
myCopy.append(3.5)
```
4.
```
myCopy = myList.copy()
myCopy.append(3.5)
```

Concept Check!

Which of the following commands will alter the value of `myList`?

```
myList = [1.0, 1.5, 2.0, 2.5, 3.0]
```

1. `myList + [3.5]` ✗
2. `myList.append(3.5)` ✓
3.
```
myCopy = myList
myCopy.append(3.5)
```
4.
```
myCopy = myList.copy()
myCopy.append(3.5)
```

Concept Check!

Which of the following commands will alter the value of `myList`?

```
myList = [1.0, 1.5, 2.0, 2.5, 3.0]
```

1. `myList + [3.5]` ✗
2. `myList.append(3.5)` ✓
3. `myCopy = myList` ✓
   `myCopy.append(3.5)`
4. `myCopy = myList.copy()`
   `myCopy.append(3.5)`

Concept Check!

Which of the following commands will alter the value of `myList`?

```
myList = [1.0, 1.5, 2.0, 2.5, 3.0]
```

1. `myList + [3.5]` ✗
2. `myList.append(3.5)` ✓
3. `myCopy = myList` ✓
   `myCopy.append(3.5)`
4. `myCopy = myList.copy()` ✗
   `myCopy.append(3.5)`

2D Lists

2D Lists

Can have a list of lists:

2D Lists

Can have a list of lists:

```
>>> myList = []
```

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
```

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

Viewed as a Table, where each sublist is a row:

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

Viewed as a Table, where each sublist is a row:

```
>>> print(myList)
```

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

Viewed as a Table, where each sublist is a row:

```
>>> print(myList)
[[0, 1],
[2, 3]]
```

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

Viewed as a Table, where each sublist is a row:

```
>>> print(myList)
[[0, 1],
[2, 3]]
```

Index using `[row][col]`:

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

Viewed as a Table, where each sublist is a row:

```
>>> print(myList)
[[0, 1],
[2, 3]]
```

Index using `[row][col]`:

```
>>> print(myList[0][1])
1
```

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

Viewed as a Table, where each sublist is a row:

```
>>> print(myList)
[[0, 1],
[2, 3]]
```

Index using `[row][col]`:

```
>>> print(myList[0][1])
1
```

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

Viewed as a Table, where each sublist is a row:

```
>>> print(myList)
[[0, 1],
[2, 3]]
```

Index using `[row][col]`:

```
>>> print(myList[0][1])
1
```

Or select a whole row:

2D Lists

Can have a list of lists:

```
>>> myList = []
>>> myList.append([0, 1])
>>> myList.append([2, 3])
```

Viewed as a Table, where each sublist is a row:

```
>>> print(myList)
[[0, 1],
[2, 3]]
```

Index using `[row][col]`:

```
>>> print(myList[0][1])
1
```

Or select a whole row:

```
>>> print(myList[0])
[0, 1]
```