List Basics

List Basics

Creating, indexing, and basic operators

List Review

List Review

We have already seen some basic usage of lists:

List Review

We have already seen some basic usage of lists:

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

List Review

We have already seen some basic usage of lists:

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

List Review

We have already seen some basic usage of lists:

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

List Review

We have already seen some basic usage of lists:

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

List Review

We have already seen some basic usage of lists:

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

List Review

We have already seen some basic usage of lists:

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

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

List Review

We have already seen some basic usage of lists:

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

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

a
e
i
o
u
```

Advanced List Creation

You can also create lists using the * operator:

Advanced List Creation

You can also create lists using the * operator:

```
>>> myList = [0] * 5
```

Advanced List Creation

You can also create lists using the * operator:

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

Advanced List Creation

You can also create lists using the * operator:

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

Advanced List Creation

You can also create lists using the * operator:

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

Or concatenation:

Advanced List Creation

You can also create lists using the * operator:

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

Or concatenation:

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

Advanced List Creation

You can also create lists using the * operator:

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

Or concatenation:

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

Advanced List Creation

You can also create lists using the * operator:

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

Or concatenation:

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

Advanced List Creation

You can also create lists using the * operator:

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

Or concatenation:

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

Basic list indexing

Basic list indexing

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

Basic list indexing

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

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

Basic list indexing

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

```
>>> vowels = ["a", "e", "i", "o", "u"]
                0
```

Basic list indexing

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

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

Basic list indexing

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

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

Basic list indexing

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

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

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
```

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])
```

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
```

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])
```

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
```

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])
```

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
```

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

>>> len(vowels)
```

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

>>> len(vowels)
5
```

The range function

The range function

This is why the `range(n)` function counts from `0, ..., n-1`

The range function

This is why the `range(n)` function counts from `0, ..., n-1`

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

The range function

This is why the `range(n)` function counts from `0, ..., n-1`

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

```
a
e
i
o
u
```

Advanced indexing

Advanced indexing

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

```
>>> vowels = ["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

Advanced indexing

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

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

$-5 \quad -4$

Advanced indexing

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

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

$-5 \quad -4 \quad -3$

Advanced indexing

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

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

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

Advanced indexing

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

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

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

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])
```

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

Slicing

Can create a sublist by "slicing" a list:

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

Slicing

Can create a sublist by "slicing" a list:

```
>>> vowels = ["a", "e", "i", "o", "u"]
>>> print(vowels[1:3])
```

Slicing

Can create a sublist by "slicing" a list:

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

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

Slicing

Can create a sublist by "slicing" a list:

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

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

>>> print(vowels[:3])
```

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]
```

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:])
```

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]
```

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[:])
```

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]
```

Concept Check!

Consider the following list:

```
myList = [1, 2, 3, 4, 5]
```

What will each of the following commands print?

1. `print(len(myList))`
2. `print(myList + [0])`
3. `print(myList[2])`
4. `print(myList[-1])`
5. `print(myList[1:-1])`
6. `print(myList[:-1])`

Concept Check!

Consider the following list:

```
myList = [1, 2, 3, 4, 5]
```

What will each of the following commands print?

1. `print(len(myList))` 5
2. `print(myList + [0])`
3. `print(myList[2])`
4. `print(myList[-1])`
5. `print(myList[1:-1])`
6. `print(myList[:-1])`

Concept Check!

Consider the following list:

```
myList = [1, 2, 3, 4, 5]
```

What will each of the following commands print?

1. `print(len(myList))` — `5`
2. `print(myList + [0])` — `[1, 2, 3, 4, 5, 0]`
3. `print(myList[2])`
4. `print(myList[-1])`
5. `print(myList[1:-1])`
6. `print(myList[:-1])`

Concept Check!

Consider the following list:

```
myList = [1, 2, 3, 4, 5]
```

What will each of the following commands print?

1. `print(len(myList))` — `5`
2. `print(myList + [0])` — `[1, 2, 3, 4, 5, 0]`
3. `print(myList[2])` — `3`
4. `print(myList[-1])`
5. `print(myList[1:-1])`
6. `print(myList[:-1])`

Concept Check!

Consider the following list:

```
myList = [1, 2, 3, 4, 5]
```

What will each of the following commands print?

1. `print(len(myList))` — `5`
2. `print(myList + [0])` — `[1, 2, 3, 4, 5, 0]`
3. `print(myList[2])` — `3`
4. `print(myList[-1])` — `5`
5. `print(myList[1:-1])`
6. `print(myList[:-1])`

Concept Check!

Consider the following list:

```
myList = [1, 2, 3, 4, 5]
```

What will each of the following commands print?

1. `print(len(myList))` — `5`
2. `print(myList + [0])` — `[1, 2, 3, 4, 5, 0]`
3. `print(myList[2])` — `3`
4. `print(myList[-1])` — `5`
5. `print(myList[1:-1])` — `[2, 3, 4]`
6. `print(myList[:-1])`

Concept Check!

Consider the following list:

```
myList = [1, 2, 3, 4, 5]
```

What will each of the following commands print?

1. `print(len(myList))` — `5`
2. `print(myList + [0])` — `[1, 2, 3, 4, 5, 0]`
3. `print(myList[2])` — `3`
4. `print(myList[-1])` — `5`
5. `print(myList[1:-1])` — `[2, 3, 4]`
6. `print(myList[:-1])` — `[1, 2, 3, 4]`

Mutability and Tuples

Mutability and Tuples

A list is *mutable*.

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

```
>>> myList = [1, 2, 3, 4]
```

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

```
>>> myList = [1, 2, 3, 4]
>>> myList[0] = 2
```

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

```
>>> myList = [1, 2, 3, 4]
>>> myList[0] = 2
```
✓

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

```
>>> myList = [1, 2, 3, 4]
>>> myList[0] = 2
```
✓

If you are just using as a named constant, then a *tuple* is *immutable*:

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

```
>>> myList = [1, 2, 3, 4]
>>> myList[0] = 2
```
✓

If you are just using as a named constant, then a *tuple* is *immutable*:

```
>>> myTuple = (1, 2, 3, 4)
```

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

```
>>> myList = [1, 2, 3, 4]
>>> myList[0] = 2
```
✓

If you are just using as a named constant, then a *tuple* is *immutable*:

```
>>> myTuple = (1, 2, 3, 4)
>>> myTuple[0] = 2
```

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

```
>>> myList = [1, 2, 3, 4]
>>> myList[0] = 2
```
✓

If you are just using as a named constant, then a *tuple* is *immutable*:

```
>>> myTuple = (1, 2, 3, 4)
>>> myTuple[0] = 2
```
✗

Mutability and Tuples

A list is *mutable*.

This means that you can alter the value of an item in a list in the future:

```
>>> myList = [1, 2, 3, 4]
>>> myList[0] = 2
```
✓

If you are just using as a named constant, then a *tuple* is *immutable*:

```
>>> myTuple = (1, 2, 3, 4)
>>> myTuple[0] = 2
TypeError:  "tuple" object does not support item
assignment
```
✗