Storing and Retrieving Data with Collections
Variables up to now

Variable: A named reference to a value in memory

\[ a = 5 \]
\[ b = 10 \]
\[ c = a + b \]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>int</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>int</td>
<td>10</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>int</td>
<td>15</td>
</tr>
</tbody>
</table>
Collections

Collection: A grouping of a variable number of values in memory

```plaintext
a = [1, 2, 3, 4, 5, 6]
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>a[0]</td>
<td>2</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>a[1]</td>
<td>3</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>a[2]</td>
<td>4</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>a[3]</td>
<td>5</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>a[4]</td>
<td>6</td>
<td>int</td>
<td>5</td>
</tr>
<tr>
<td>a[5]</td>
<td>7</td>
<td>int</td>
<td>6</td>
</tr>
</tbody>
</table>
Collections

Collection: A grouping of a variable number of values in memory

```python
a = [1, 2, 3, 4, 5, 6]
a[0] = 0
a[1] = -1
print(a)
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>a[0]</td>
<td>2</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>a[1]</td>
<td>3</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>a[2]</td>
<td>4</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>a[3]</td>
<td>5</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>a[4]</td>
<td>6</td>
<td>int</td>
<td>5</td>
</tr>
<tr>
<td>a[5]</td>
<td>7</td>
<td>int</td>
<td>6</td>
</tr>
</tbody>
</table>
Collections

Collection: A grouping of a variable number of values in memory

```python
a = [1, 2, 3, 4, 5, 6]
a[0] = 0
a[1] = -1
print(a)
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>a[0]</td>
<td>8</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>a[1]</td>
<td>3</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>a[2]</td>
<td>4</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>a[3]</td>
<td>5</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>a[4]</td>
<td>6</td>
<td>int</td>
<td>5</td>
</tr>
<tr>
<td>a[5]</td>
<td>7</td>
<td>int</td>
<td>6</td>
</tr>
</tbody>
</table>
Collections

Collection: A grouping of a variable number of values in memory

```python
a = [1, 2, 3, 4, 5, 6]
a[0] = 0
a[1] = -1
print(a)
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>a[0]</td>
<td>8</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>a[1]</td>
<td>9</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>a[2]</td>
<td>4</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>a[3]</td>
<td>5</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>a[4]</td>
<td>6</td>
<td>int</td>
<td>5</td>
</tr>
<tr>
<td>a[5]</td>
<td>7</td>
<td>int</td>
<td>6</td>
</tr>
</tbody>
</table>
def main():
a = 1
b = a
b += 1
print(a)
print(b)

if __name__ == "__main__":
    main()
def main():
    a = 1
    b = a
    b += 1
    print(a)
    print(b)

if __name__ == "__main__":
    main()
def main():
a = 1
b = a
b += 1
print(a)
print(b)

if __name__ == "__main__":
    main()
Mutable Data Types

def main():
a = [1,2,3,4]
b = a
b[0] += 1
print(a)
print(b)

if __name__ == "__main__":
    main()
Mutable Data Types

def main():
a = [1,2,3,4]
b = a
b[0] += 1
print(a)
print(b)

if __name__ == "__main__":
    main()
def main():
    a = [1,2,3,4]
    b = a
    b[0] += 1
    print(a)
    print(b)

if __name__ == "__main__":
    main()
Mutable Data Types

def main():
a = [1, 2, 3, 4]
b = a
b[0] += 1
print(a)
print(b)

if __name__ == "__main__":
    main()
def add1(s):
    s += 1
    return s

def main():
    a = 5
    b = add1(a)
    print(a)

if __name__ == "__main__":
    main()
Mutability vs Immutability

def add1(s):
    s[0] -= 1
    return s

def main():
    a = [1, 2, 3, 4]
    b = add1(a)
    print(a)
    print(a)

if __name__ == "__main__":
    main()
def add1(s):
    s[0] -= 1
    return s

def main():
    a = [1, 2, 3, 4]
    b = add1(a)
    print(a)

    if __name__ == '__main__':
        main()
def add1(s):
    s[0] -= 1
    return s

def main():
    a = [1, 2, 3, 4]
    b = add1(a)
    print(a)

if __name__ == "__main__":
    main()
def add1(s):
    s[0] -= 1
    return s

def main():
    a = [1,2,3,4]
    b = add1(a)
    print(a)

if __name__ == "__main__":
    main()
def add1(s):
    s[0] -= 1
    return s

def main():
    a = [1, 2, 3, 4]
    b = add1(a)
    print(a)

if __name__ == "__main__":
    main()
Copy Collections if You Want to Preserve Changes

def main():
    a = [1,2,3,4]
    b = list(a)
    b[0] -= 1
    print(a)
    print(b)

if __name__ == "__main__":
    main()
Copy Collections if You Want to Preserve Changes

def main():
    a = [1, 2, 3, 4]
    b = list(a)
    b[0] -= 1
    print(a)
    print(b)

if __name__ == "__main__":
    main()
Copy Collections if You Want to Preserve Changes

def main():
    a = [1, 2, 3, 4]
    b = list(a)
    b[0] -= 1
    print(a)
    print(b)

if __name__ == "__main__":
    main()
Copy Collections if You Want to Preserve Changes

def main():
    a = [1,2,3,4]
    b = list(a)
    b[0] -= 1
    print(a)
    print(b)

if __name__ == "__main__":
    main()
Tuple

- Tuples are immutable and do not support modification or resizing
- Can access items using indexing
- Convert any collection to a tuple using the tuple() function

```python
a = (1, 2, 3)
b = a[0]
c = a[1]
d = a[2]
b, c, d = a
```
Lists

- Using indexing to get individual items from a list
- Resize lists using append()
- Use list() to convert a collection to a list

```python
a = [1, 2, 3]
b = a[0]
a[0] = -1
a.append(4)
```
String/List/Tuple Indexing

```
a = list(range(10,16))
```
String/List/Tuple Indexing

```python
a = list(range(10,16))
```

- `a[index]`  
- `a[0]`  
- `a[5]`  
- `a[-1]`
String/List/Tuple Indexing

```python
a = list(range(10,16))
```

```
a[0]
a[5]
a[-1]
```

```
Index 0 1 2 3 4 5
```

```
a 10 11 12 13 14 15
```

The last item in the list is always `len() - 1`
String/List/Tuple Indexing

```python
a = list(range(10,16))

a[0]  # a[0]

a[-1]  # a[-1]
```

Index

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>

Negative Index

<table>
<thead>
<tr>
<th>-6</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
</tr>
</thead>
</table>

The last item in the list can be found with a negative index.
String/List/Tuple Indexing

```
a = list(range(10,16))

a[index]

a[start: stop]

b = a[3:]
b = a[1: 4]
b = a[ : 3]
```

If no stop index is given, it’s implied the stop index is the end of the list.
String/List/Tuple Indexing

```python
a = list(range(10, 16))
```

Indexing with a range includes up to, but not including the stop index.
String/List/Tuple Indexing

```python
a = list(range(10,16))
```

- `a[index]`
- `a[start: stop]`
  - `a[3:]`
  - `a[1: 4]`
  - `a[: 3]`

Index Table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If no start index is given, it's implied the start index is 0.
String/List/Tuple Indexing

\[ a = \text{list}(\text{range}(10, 16)) \]

\[ a[\text{start: stop: step}] \]
\[ b = a[::2] \]
\[ b = a[1:5:3] \]
\[ b = a[:::-1] \]
String/List/Tuple Indexing

```python
a = list(range(10,16))
print(a)
print(a[::2])
print(a[1: 5: 3])
print(a[::-1])
```

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>

```
```

```python
b = a[1: 5: 3]
print(b)
```

```
```

```python
print(a[::-1])
```

```
```

```python
print(11, 14)
```

```
```
String/List/Tuple Indexing

a = list(range(10,16))

a[start: stop: step]
b = a[::2]
b = a[1: 5: 3]
b = a[::-1]

A negative step will reverse the order
How to check a palindrome?

```python
sentence = "Engage le jeu que je le gagne"
sentence = sentence.replace(' ', '').lower()
if sentence == sentence[::-1]:
    print(sentence + " is a palindrome?")
```

```python
sentence = "Engage le jeu que je le gagne"
sentence = sentence.replace(' ', '').lower()
if sentence == sentence[:::-1]:
    print(sentence + " is a palindrome?")
```
List Operations

List Concatenation

\[ a = [1,2,3] \]
\[ b = [4,5,6] \]
\[ c = a + b \]

List Multiplication

\[ a = [0,] \times 5 \]
Sets

a = {1, 2, 3}
a.add(4)
if 2 in a:
    print(a)

• A set is an unordered collection of unique items

• Collections which don’t preserve the order in which elements are added are called unordered

• Doesn’t support indexing
Indexing into Sets

- Indexing into sets is not possible, if you want to only get one item, or a few you must convert it into a list
Set Operations

Set Union: A set containing items from both sets

\[ a = \{1, 2, 3\} \cup \{3, 4\} \]

Set Intersection: A set containing items found only in both sets

\[ a = \{1, 2, 3\} \cap \{3, 4\} \]

Set Difference: The elements that exist in one set, but not the other

\[ a = \{1, 2, 3\} - \{3, 4\} \]
Dictionaries

a = {'a':1, 'b':2, 'c':3}
a['a'] = 4
if 'a' in a:
    print(a['a'])

• A Dictionary is an unordered collection of key-value pairs
• Keys should be all the same type, but it’s not enforced by Python
Accessing values in collection

- If you get an item from a tuple or a list using indexing or from a dictionary using a key, you need to check it’s a valid index or key

```python
a = [1,2,3]
if index >= 0 and index < len(a):
    print(a[index])

a = {'a':1, 'b':2, 'c':3}
if key in a:
    print(a[key])
```
When to use Lists, Sets, Dictionaries

- Use lists when you can, they are generally the fastest
- Also use lists when your items need to be kept in order (like a sorted order)
- Use sets if you need your collection to only contain unique items
- Use sets if you are going to do a lot of ‘in’ checks
- Use dictionaries if you need to associate a value with a key (a number with a word)
- ‘in’ checks with dictionaries is also as fast as with a set