CSCI 141
Computer Programming I

Filip Jagodzinski
From Last Time

myList = ["hello", 2.0, 5, [10, 20]]

What is the reference diagram for this code?
From Last Time

```python
myList = ["hello", 2.0, 5, [10, 20]]
```
From Last Time

```
myList = ["hello", 2.0, 5, [10, 20]]
```

```
print(myList[1])
print(myList[3])
if "hi" in myList:
    print("hi is in myList")
print(len(myList))
```

Q: What is the output of the program on the left?
myList = ["hello", 2.0, 5, [10, 20]]

Q: What is the output of the program on the left?

```
print(myList[1])
print(myList[3])
if "hi" in myList:
    print("hi is in myList")
print(len(myList))
```
Task: Explain how the list `myList` is modified when the below code is executed

```
myList[0] = "Fred"
myList.append("Susan")
```
Q: What is the output of the program on the left?

```
print(len(myList))
print(myList[0:4])
```
From Last Time

```python
print(len(myList))
print(myList[0:4])
```

```python
5
['Fred', 2.0, 5, [10, 20]]
```
Warmup

Q: What is the output of the below program?

```python
myList = ["An entry", "another", 23]
myList[0] = "Fred"
myList.append(44)
del myList[0:2]
myList.append(44)
print(myList[1:len(myList)])
```
Warmup

Q: What is the output of the below program?

```python
myList = ["An entry", "another", 23]
myList[0] = "Fred"
myList.append(44)
del myList[0:2]
myList.append(44)
print(myList[1:len(myList)])
```

myList = [“An entry”, “another”, 23]
Q: What is the output of the below program?

```python
myList = ["An entry", "another", 23]
myList[0] = "Fred"
myList.append(44)
del myList[0:2]
myList.append(44)
print(myList[1:len(myList)])
```

myList = ["An entry", "another", 23]
myList = ["Fred", "another", 23]
Warmup

Q: What is the output of the below program?

```python
myList = ["An entry", "another", 23]
myList[0] = "Fred"
myList.append(44)
del myList[0:2]
myList.append(44)
print(myList[1:len(myList)])
```

myList = ["An entry", "another", 23]
myList = ["Fred", "another", 23]
myList = ["Fred", "another", 23, 44]
Q: What is the output of the below program?

```python
myList = ["An entry", "another", 23]
myList[0] = "Fred"
myList.append(44)
```
Q: What is the output of the below program?

```python
myList = ["An entry", "another", 23]
myList[0] = "Fred"
myList.append(44)
del myList[0:2]
myList.append(44)
print(myList[1:len(myList)])
```

myList = [“An entry”, “another”, 23]
myList = [“Fred”, “another”, 23]
myList = [“Fred”, “another”, 23, 44]
myList = [23, 44]
myList = [23, 44, 44]
Warmup

Q: What is the output of the below program?

```python
myList = ["An entry", "another", 23]
myList[0] = "Fred"
myList.append(44)
del myList[0:2]
myList.append(44)
print(myList[1:len(myList)])
```

myList = ["An entry", "another", 23]
myList = ["Fred", "another", 23]
myList = ["Fred", "another", 23, 44]
myList = [23, 44]
myList = [23, 44, 44]

Because `len(myList)` returns 3

```python
print (myList[1:3])
```

```python
[44, 44]
```
Today

Sublists
The `is` operator
Objects and References

You are nearing the completion of CSCI141. The concept of objects, although not explicitly explained until CSCI145, is a very important concept.

Nearly ALL code written for industry and “in the real world” is object oriented.

This lecture focuses on several String and List principles that are best (easiest) to explain if you think of Strings and Lists as objects.
Task: Write python code that creates the below list. Name the list `randomItems`.
Lists, subLists

randomItems = ["WWU", -17, "bham", 22, 34.0, 674]
Q: What is the python code that sums all (including sublist) integer elements?
Lists, subLists

As a first step, identify the integers
As a first step, identify the integers.
Does the following python code sum the integers? Why or why not?

```python
```
We need the ability to access a sublist’s entries

Q: How is the value 22 accessed?
Lists, subLists

```
print(randomItems[2][1])
```
Lists, subLists

The `print` function ... we’ve seen this before
Lists, subLists

randomItems is the reference variable assigned to refer to the list
Lists, subLists

index 2 of randomItems refers to the element as position 2, which is a list

```python
print(randomItems[2][1])
```
Lists, subLists

index 1 of the list that is the second element of randomItems refers to 22

print(randomItems[2][1])

index 1 of the list that is the second element of randomItems refers to 22
Q: What is the python code that sums all (including sublist) integer elements?
Q: What is the python code that sums all (including sublist) integer elements?

```python
```
Q: What is the python code that sums all (including sublist) integer elements?

```python
```
Assume you have access to `randomItems`, but you do not know its contents (hence each element can be a string, a float, an integer, or another list). Further assume that if an element of `randomItems` is a list (hence a sublist), then that sublist CANNOT have elements that themselves are sublists*.

*This second assumption is for demonstration purposes only. You can create lists that have elements that are lists, whose elements are lists, whose elements are lists ...
Assume you have access to `randomItems`, but you do not know its contents (hence each element can be a string, a float, an integer, or another list). Further assume that if an element of `randomItems` is a list (hence a sublist), then that sublist CANNOT have elements that themselves are sublists*

*This second assumption is for demonstration purposes only. You can create lists that have elements that are lists, whose elements are lists, whose elements are lists ...
Assume you have access to `randomItems`, but you do not know its contents (hence each element can be a string, a float, an integer, or another list). Further assume that if an element of `randomItems` is a list (hence a sublist), then that sublist CANNOT have elements that themselves are sublists*.

*This second assumption is for demonstration purposes only. You can create lists that have elements that are lists, whose elements are lists, whose elements are lists ...
Lists, subLists

Assume you have access to `randomItems`, but you do not know its contents (hence each element can be a string, a float, an integer, or another list). Further assume that if an element of `randomItems` is a list (hence a sublist), then that sublist CANNOT have elements that themselves are sublists.*

*This second assumption is for demonstration purposes only. You can create lists that have elements that are lists, whose elements are lists, whose elements are lists …

Task home exercise: write Python code that sums all (including sublist) integer elements of `randomItems`.

In order to write code to accomplish this task, in addition to being able to access a sublist’s entries, what other capability must you have?
Lists, subLists

Assume you have access to `randomItems`, but you do not know its contents (hence each element can be a string, a float, an integer, or another list). Further assume that if an element of `randomItems` is a list (hence a sublist), then that sublist CANNOT have elements that themselves are sublists*.

*This second assumption is for demonstration purposes only. You can create lists that have elements that are lists, whose elements are lists, whose elements are lists ...

Task home exercise: write Python code that sums all (including sublist integer elements of `randomItems`).

You must be able to determine the type of each element. We’ve already seen use of the `type` function.

Q: How is `type` used to infer the type of an object?
Lists, subLists

type(randomItems[1]) is int
Lists, subLists

randomItems

?  ?  ?  ?

\[
type(randomItems[1]) \text{ is int}
\]

is is a binary operator.
is is a binary operator ... if both operands are data types, it returns True if both operands are the same type, and it returns False otherwise.
Lists, subLists

Assume you have access to `randomItems`, but you do not know its contents (hence each element can be a string, a float, an integer, or another list). Further assume that if an element of `randomItems` is a list (hence a sublist), then that sublist CANNOT have elements that themselves are sublists*.

*This second assumption is for demonstration purposes only. You can create lists that have elements that are lists, whose elements are lists, whose elements are lists ...

Task home exercise: write Python code that sums all (including sublist) integer elements of `randomItems`.

You’ll need to use `is`, `len`, `for`, `if`, `type`, and possibly `range`.

Hint: The above task can be coded with 10 or fewer lines of code.
Objects and references

```
aVar = "dog"
anotherVar = "dog"
```

We’ve seen this before ... what do the above two lines of code accomplish?
Objects and references

```
aVar = "dog"
anotherVar = "dog"
```

String

```
"dog"
```

String

```
"dog"
```

Is this what python does?
Objects and references

Remember that strings are immutable, so once created, they cannot be modified, so to save space ...
Objects and references

Remember that strings are immutable, so once created, they cannot be modified, so to save space ...

If a new object of type String is created, and it has the contents of an already created String object, then the new variable will be assigned to the old object
Objects and references

```java
 aVar = "dog"
 anotherVar = "dog"
```

The `is` operator, if both operands are variables, returns `True` if the two variable references (variables to objects) refer to the same object. Else it returns `False`. Thus when discussing strings:

```java
 if (aVar is anotherVar)
   evaluates to True
```

Remember that strings are immutable, so once created, they cannot be modified, so to save space ...

If a new object of type String is created, and it has the contents of an already created String object, then the new variable will be assigned to the old object.
Objects and references

With lists, we must be a bit more careful when using is or ==

Q: Do is and == perform the same function?
Objects and references

With lists, we must be a bit more careful when using is or ==

Q: Do is and == perform the same function?

```python
myList = ["a", 32]
anotherList = ["a", 32]
```

Task: Draw the diagram that describes the relationship between the variables and the lists that they refer to.
Objects and references

With lists, we must be a bit more careful when using `is` or `==`

Q: Do `is` and `==` perform the same function?

Here, `myList` is a reference to a collection of references

```python
myList = ["a", 32]
anotherList = ["a", 32]
```
Objects and references

With lists, we must be a bit more careful when using `is` or `==`

Q: Do `is` and `==` perform the same function?

Here, `myList` is a reference to a collection of references, which is a DIFFERENT collection than `anotherList`, which is a collection of references.
Objects and references

With lists, we must be a bit more careful when using `is` or `==`

Q: Do `is` and `==` perform the same function?

Here, `myList` is a reference to a collection of references, which is a DIFFERENT collection than `anotherList`, which is a collection of references.

Because Strings are immutable, if two String variables refer to the SAME characters, then the two String variables point to the SAME String object.
Objects and references

With lists, we must be a bit more careful when using `is` or `==`.

**Q: Do `is` and `==` perform the same function?**

Here, `myList` is a reference to a collection of references, which is a DIFFERENT collection than `anotherList`, which is a collection of references.

Because Strings are immutable, if two String variables refer to the SAME characters, then the two String variables point to the SAME String object.

Because Lists are mutable, if two List variables contain the same elements, then the two List variables do NOT refer to the same list objects.
With lists, we must be a bit more careful when using `is` or `==`.

Q: Do `is` and `==` perform the same function?

Here, `myList` is a reference to a collection of references, which is a DIFFERENT collection than `anotherList`, which is a collection of references.

Q: What is the output of the following code:

```python
print(myList is anotherList)
print(myList == anotherList)
```
Objects and references

With lists, we must be a bit more careful when using `is` or `==`.

Q: Do `is` and `==` perform the same function?

Here, `myList` is a reference to a collection of references, which is a DIFFERENT collection than `anotherList`, which is a collection of references.

Q: What is the output of the following code:

```python
print(myList is anotherList)
print(myList == anotherList)
```

False
True
Q: Is it possible to have two variables of type `List` refer to the SAME list object?
Objects and references

\[
a = [3, 4, 5] \\
b = a
\]

Task: Draw the reference diagram for the above code
Objects and references

\[
a = [3, 4, 5] \\
b = a
\]
Objects and references

With this approach you must be careful because ANY change that you make to \( \text{a} \) will also “change” \( \text{b} \) because they both point to the same COLLECTION of references to objects.
Objects and references

With this approach you must be careful because ANY change that you make to \( a \) will also “change” \( b \) because they both point to the same COLLECTION of references to objects.

Q: What is the output of the below code?

```python
a=[3,4,5]
b=a
a[2] = 7
print(b)
```

live demo
Objects and references

How can you make a “real” copy (clone) of a list, so that the new and old (cloned) list point to different objects?
Objects and references

```python
a = [3, 4, 5]
b = a[:]
```

Use the slice operator without no operands

How can you make a “real” copy (clone) of a list, so that the new and old (cloned) list point to different objects?
Objects and references

Use the slice operator without no operands

Q: What is the output of the below code?

```
[a = [3, 4, 5]
b = a[:]
```

How can you make a “real” copy (clone) of a list, so that the new and old (cloned) list point to Different objects?

```
a=[3,4,5]
b = a[:]
a[2] = 7
print(b)
```

live demo
Up Next

List repetition
List concatenation
Functions that use/modify lists