CSCI 141
Computer Programming I

Filip Jagodzinski
Announcements

- **Project**
  - Pseudocode is due tonight, Monday.
    - Submit enough pseudocode to demonstrate that you’ve started thinking about the high-level structure of your program.

- **Schedule for remainder of Quarter**
  - Today: last “real” lecture; course evaluations
  - Tuesday and Wednesday, 8 and 9 March: no lab; work on your final project
  - Wednesday, 9 March: sample final exam solutions presented in lecture
  - Friday, 11 March: no lecture
  - Sunday, 13 March: extra review session, CF125, 5-6pm

- **Extra Credit**
  - Submit a single question and the answer for 1% extra credit that will be added to your final course percentage.
  - I’ll distribute all submitted questions publicly available which you can use for additional prep for the final exam
  - One of the submitted questions will appear on the final exam
From Last Time

Appending modifies an existing list, while concatenation creates a NEW list.

```python
aList = ["almost done", 23]
aList.append(34.55)
print(aList)
```

This pieces of code results in ...
From Last Time

Appending modifies an existing list, while concatenation creates a NEW list.

```
aList = ["almost done", 23]
aList.append(34.55)
print(aList)
```

```
aList = ["almost done", 23]
newList = aList + [34.55]
print(newList)
```

This pieces of code results in ...

```
newList and aList refer to different list objects, but the elements of those two objects are the same.
```
The opposite of `split` is to `join`. Just like `split` needs to know about the delimiter (by default it is the empty space), when you `join` 2 items, you must specify what is the “glue” that will hold two things together when they are joined.

```python
words = ["spring", "is", "here"]
words.append("almost")
print("RAIN".join(words))
```

Q: What is the output of the above program?

Live demo
Today

Dictionaries
Wrapping up
Course Evaluations
Task: write a python program that will keep track of how many coins, books, plants, and bills you have. The program should print the total count of items.
Dictionaries

Task: write a python program that will keep track of how many coins, books, plants, and bills you have. The program should print the total count of items.

One approach is to declare as many variables as are needed ...

numCoins = 7
numBooks = 2
numPlants = 4
numBills = 5

Q: How do you then print out the sum of items?
Dictionaries

Task: write a python program that will keep track of how many coins, books, plants, and bills you have. The program should print the total count of items.

That is annoyingly annoying and besides, so much busy work! Imagine trying to sum 300 items. There has to be a better way.

Q: How do you then print out the sum of items?

print("Total items: " + str(numCoins + numBooks + numPlants + numBills))

Q: What is another way?

One approach is to declare as many variables as are needed ...

numCoins = 7
numBooks = 2
numPlants = 4
numBills = 5
Dictionaries

Task: write a python program that will keep track of how many coins, books, plants, and bills you have. The program should print the total count of items.

Another approach is to declare a list

```python
myItems = [7, 2, 4, 5]
```

Q: How do you then print out the sum of items?

Any ideas? Live demo
Task: write a python program that will keep track of how many coins, books, plants, and bills you have. The program should print the total count of items.

This is better, and neater. But what disadvantage does this approach have?

Another approach is to declare a list

```python
myItems = [7, 2, 4, 5]
```

Q: How do you then print out the sum of items?

```python
numItems = 0
for item in myItems:
    numItems = numItems + int(item)
print(numItems)
```

This is better, and neater. But what disadvantage does this approach have?
Dictionaries

Task: write a python program that will keep track of how many coins, books, plants, and bills you have. The program should print the total count of items.

This is better, and neater. But what disadvantage does this approach have?

Another approach is to declare a list

myItems = [7, 2, 4, 5]

Q: How do you then print out the sum of items?

numItems = 0
for item in myItems:
    numItems = numItems + int(item)
print(numItems)

This is better, and neater.

You have to remember that the entry at index 0 refers to coins, the integer at index 1 refers to book, etc.

Q: Can you think of yet another approach? (think back to the last lab)
Dictionaries

Task: write a python program that will keep track of how many coins, books, plants, and bills you have. The program should print the total count of items.

Create a List of sublists

Q: What disadvantage does this approach have?

myItems = [["books",2], ["coins",7], ["bills",5], ["plants",4]]
numItems = 0
for x in range(0,len(myItems)):
    numItems = numItems + myItems[x][1]
print(numItems)
Dictionaries

All of these work, but for different reasons none of them are ideal.

We want an easy way to associate, or **map**, a word to a number. Abstractly, we want to map one object (in our case words) to another unique object (in our case a number)

```python
myItems = ["books", 2, "coins", 7, "bills", 5, "plants", 4]
numItems = 0
for item in myItems:
    numItems = numItems + int(item)
print(numItems)
```

```python
numCoins = 7
numBooks = 2
numPlants = 4
numBills = 5
print("Total items: " + str(numCoins + numBooks + numPlants + numBills))
```

```python
numCoins = 7
numBooks = 2
numPlants = 4
numBills = 5
print("Total items: " + str(numCoins + numBooks + numPlants + numBills))
```
We want to retain the associations that exist between the words and numbers ...
We want to retain the associations that exist between the words and numbers ...
Dictionaries

We want to retain the associations that exist between the words and numbers ...

... without all of the other clutter
We want to retain the associations that exist between the words and numbers ...

... without all of the other clutter

We want to map these “words” to the numbers with which they are associated.

“coins” -> 7

“plants” -> 4

“books” -> 2

“bills” -> 5
Dictionaries

We want to retain the associations that exist between the words and numbers ...

A dictionary is Python’s mapping type.
We want to retain the associations that exist between the words and numbers ...

A dictionary is Python’s mapping type.

To create the dictionary items
We want to retain the associations that exist between the words and numbers ...

A **dictionary** is Python’s mapping type.

To create the dictionary **items**

Create an empty dictionary, here naming it **items**, denoted by curly brackets.

```python
items = {}
```
Dictionaries

We want to retain the associations that exist between the words and numbers ...

A dictionary is Python’s mapping type.

To create the dictionary `items`

Create an empty dictionary, here naming it `items`, denoted by curly brackets, and then add key-value pairs.

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
```

Each key must be unique. However more than one key may have the same value.
We want to retain the associations that exist between the words and numbers ...

A **dictionary** is Python’s mapping type.

To create the dictionary `items`

Create an empty dictionary, here naming it `items`, denoted by curly brackets, and then add key-value pairs.

```
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
```

Ordering does not matter! Python takes care of the behind-the-scenes implementation details.
We want to retain the associations that exist between the words and numbers ...

Q: What is the output of the code on the right?

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
print(items)
```
We want to retain the associations that exist between the words and numbers ...

Q: What is the output of the code on the right?

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
print(items)
```

```python
{'books': 2, 'coins': 7, 'bills': 5, 'plants': 4}
```
Dictionaries

We want to retain the associations that exist between the words and numbers ...

```
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
print(items)
```

Either of these two accomplish the same thing. The bottom is the short-hand version

```
items = {"coins":7,"books":2,"plants":4,"bills":5}
print(items)
```
We want to retain the associations that exist between the words and numbers ...

```
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
print(items)
```

```
myString = "an apple"
print(myString[4])
twoLetters = ["A","B"]
print(twoLetters[1])
```

Q: Any guesses how we might print the value associated with the key *bills*
We want to retain the associations that exist between the words and numbers ...

Retrieving the value for a key is similar to how individual characters in a string, or elements in a list, are accessed.

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
print(items["bills"])
```

Q: Any guesses how we might print the value associated with the key `bills`
Dictionaries

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
```

Just as there are many functions for accessing and performing actions on Strings and Lists, there are many functions for modifying and accessing dictionary key-value pairs

These include

```python
del
keys()
values()
get()
in
```
Dictionaries

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
```

Just as there are many functions for accessing and performing actions on Strings and Lists, there are many functions for modifying and accessing dictionary key-value pairs.

These include:

- `del`  
- `keys()`  
- `values()`  
- `get()`  
- `in`

```python
del items["plants"]
```

This would delete the key-value `plants` from the dictionary `items`.
Dictionaries

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
```

Just as there are many functions for accessing and performing actions on Strings and Lists, there are many functions for modifying and accessing dictionary key-value pairs.

These include:

- `del`  
- `keys()`  
- `values()`  
- `get()`  
- `in`

```python
# Example usage
print(items.keys())
```

```python
# Example usage
del items["plants"]
```

This would print all of the keys in `items`.
Just as there are many functions for accessing and performing actions on Strings and Lists, there are many functions for modifying and accessing dictionary key-value pairs. These include:

- `del`: Remove an item from the dictionary.
- `keys()`: Return a new view of the dictionary's keys.
- `values()`: Return a new view of the dictionary's values.
- `get()`: Return the value for a key, or a default value if the key is not found.
- `in`: Check if a key is in the dictionary.

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5

# Remove "plants" from items
del items["plants"]
# Print all keys in items
print(items.keys())
# Print all values in items
print(items.values())
```

This would print all of the values for all keys in `items`. From the API:

- `values()`: Return a copy of the dictionary's list of values. See the note for `dict.items()`. 
Dictionaries

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
```

Just as there are many functions for accessing and performing actions on Strings and Lists, there are many functions for modifying and accessing dictionary key-value pairs.

These include:

- `del`
- `keys()`
- `values()`
- `get()`
- `in`

The `get` method returns a value associated with a specific key. This would print 4, because that is the value associated with the key "plants"
Dictionaries

```
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
```

Just as there are many functions for accessing and performing actions on Strings and Lists, there are many functions for modifying and accessing dictionary key-value pairs.

These include

- `del`
- `keys()`
- `values()`
- `get()`
- `in`

```
del items["plants"]
print(items.keys())
print(items.values())
print(items.get("plants"))
print("dog" in items)
```

The `in` operator checks if the left operand is a key in the dictionary specified by the right operand. In this case, the output would be `False` because `dog` is not a key in the dictionary `items`. 
Dictionaries

```python
items = {}
items["coins"] = 7
items["books"] = 2
items["plants"] = 4
items["bills"] = 5
```

Returning to the original task:

Write a python program that sums the number of items in the dictionary `items`

```python
del keys()
values()
```
Q: What is the output of this program?

```python
 aVar = 32
 anotherVar = "76"
 # This is a comment
 print("1. " + anotherVar[-2:-1])
 aCalculation = aVar + int(anotherVar)
 if (aCalculation > 50):
   print("2. The result is", aCalculation)
 for x in range(0, len(anotherVar)):
   print("3. " + anotherVar[x] * 3)
 def myCustomFunction(param1):
   return param1[0]
 print("4. " + myCustomFunction(anotherVar))
 myList = [anotherVar[0], anotherVar[1]]
 aNewCopy = myList[:]
 myDictionary = {"firstL":"7", "secondL":"6"}
 theLetters = ""
 for x in myDictionary.values():
   theLetters = theLetters + x
 print("5. " + theLetters)
```
Q: What is the output of this program?

```python
 aVar = 32
 anotherVar = "76"
 # This is a comment
 print("1. " + anotherVar[-2:-1])
 aCalculation = aVar + int(anotherVar)
 if (aCalculation > 50):
     print("2. The result is", aCalculation)
 for x in range(0, len(anotherVar)):
     print("3. " + anotherVar[x] * 3)
 def myCustomFunction(param1):
     return param1[0]
 print("4. " + myCustomFunction(anotherVar))
 myList = [anotherVar[0], anotherVar[1]]
 aNewCopy = myList[:]
 myDictionary = {"firstL":"7", "secondL":"6"}
 theLetters = ""
 for x in myDictionary.values():
     theLetters = theLetters + x
 print("5. " + theLetters)
```

Declaring a variable

This is how we write programs to “remember” data

The assignment operator is a binary operator, and thus requires two operands
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"

# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7

print is a function

when both operands to + are strings, then the + is the concatenation operator

the : permits you to slice a String

both positive and “negative” indices can be used to specify characters in a string
Q: What is the output of this program?

```python
 aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])

aCalculation = aVar + int(anotherVar)

if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7

when both operands to + are integers, then the + is the addition operator

`int` is a conversion function
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0,len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7

*if* allows you to write a selection statement

The : at the end of this line is required

A boolean expression evaluates to either True or False
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108

The body of a if statement is evaluated ONLY if the condition evaluates to True

One or more statements may be "in the body" of the if statement
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108

*for* allows us to iterate ... to perform a task many times

*range* is a function that can be invoked with one, two, or three arguments

*len* is a function that returns the length of the argument
```
Q: What is the output of this program?

```avar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)

def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0,len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":7, "secondL":6}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108
3. 777
4. 666

For calculations and/or procedures that you perform often, you should declare a custom function

This is the declaration of the function `myCustomFunction`, with a single parameter

The function returns the item at index 0 of local parameter `param1`
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0,len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108
3. 777
4. 7

The function `myCustomFunction` returns the String "7" upon input of the reference variable `anotherVar`, which refers to the String object that has value "76"
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108
3. 777
4. 7

Lists are made up of elements

This variable `myList` refers to a 2-element list, where the element at index 0 is the String "7" and the element at index 1 is the String "6"
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108
3. 777
4. 7

To make a copy of a list (top-list only) use the slice operator

`aNewCopy` is a variable that refers to a newly created object of type `List` that has the same elements as `myList`
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108
3. 777
4. 7

Dictionaries are collections of key-value pairs

The dictionary `myDictionary` has two key-value pairs, they are

"firstL" → "7"
"secondL" → "6"
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0, len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108
3. 777
4. 7

The variable `theLetters` is referring to String object that is the empty string
Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0,len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL" : "7", "secondL" : "6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108
3. 777
4. 666
5. 7

`values()` returns a list of the values in the dictionary `myDictionary`

Used in combination with a for loop, the `x` iterator variable assumes each element in the list, one by one, each time appending to the String `theLetters`
5 Minute review of CSCI141

Q: What is the output of this program?

```python
aVar = 32
anotherVar = "76"
# This is a comment
print("1. " + anotherVar[-2:-1])
aCalculation = aVar + int(anotherVar)
if (aCalculation > 50):
    print("2. The result is", aCalculation)
for x in range(0,len(anotherVar)):
    print("3. " + anotherVar[x] * 3)
def myCustomFunction(param1):
    return param1[0]
print("4. " + myCustomFunction(anotherVar))
myList = [anotherVar[0], anotherVar[1]]
aNewCopy = myList[:]
myDictionary = {"firstL":"7", "secondL":"6"}
theLetters = ""
for x in myDictionary.values():
    theLetters = theLetters + x
print("5. " + theLetters)
```

1. 7
2. The result is 108
3. 777
4. 7
5. 76

The String theLetters has been appended to twice, once from the empty string "" to "7", then from "7" to "76"
Up next

Class evaluations

On Wed: Solutions to sample final exam