CSCI 141
Computer Programming I

Filip Jagodzinski
Announcements

• Details of the final project have been posted to the course website
  • Three submission dates
  • Open-ended
  • Final code due at the end of the last week of classes

• Several students have asked me how “better” to prepare for the final exam … comments

• The remainder of the course …
From Last Time

```python
def performCalc(base, exponent, thirdNum):
    print(base ** exponent - thirdNum)

performCalc(1, 4, 5)
performCalc(2, -3, 6)
performCalc(3, 7, 8)
performCalc(4, 3, 2)
performCalc(8, 2, 3)
```

Now that you’ve used and written your own functions, HOW is the function `performCalc` invoked each time?

Task: Be able to explain how this code “works”
From Last Time

```python
def performCalc(base, exponent, thirdNum):
    print(base ** exponent - thirdNum)
```

```
performCalc(1, 4, 5)
performCalc(2, -3, 6)
performCalc(3, 7, 8)
performCalc(4, 3, 2)
performCalc(8, 2, 3)
```

Code in python is “processed” by the interpreter top-down

As a first step, the function `performCalc` is declared. It is **NOT** executed. Think of this as the interpreter finding out what function is available, so that if it is needed in the future the computer will know what to do.
From Last Time

```python
def performCalc(base, exponent, thirdNum):
    print(base ** exponent - thirdNum)

performCalc(1, 4, 5)
performCalc(2, -3, 6)
performCalc(3, 7, 8)
performCalc(4, 3, 2)
performCalc(8, 2, 3)
```

The function `performCalc` is executed with the first argument being the number 1, the second argument being the number 4, and third argument the number 5, in that order.
From Last Time

The function `performCalc` is executed with the first argument being the number 1, the second argument being the number 4, and third argument the number 5, in that order.

The function has already been declared, and the computer “executes” the function with the indicated arguments.
From Last Time

The function `performCalc` is executed with the first argument being the number 1, the second argument being the number 4, and third argument the number 5.

In the declaration, it specifics that...

- the first argument is assigned to the first parameter, `base`.
- the second argument is assigned to the second parameter, `exponent`.
- the third argument is assigned to the third parameter, `thirdNum`.

```python
def performCalc(base, exponent, thirdNum):
    print(base ** exponent - thirdNum)

performCalc(1, 4, 5)
performCalc(2, -3, 6)
performCalc(3, 7, 8)
performCalc(4, 3, 2)
performCalc(8, 2, 3)
```
From Last Time

The function `performCalc` is executed with the first argument being the number 1, the second argument being the number 4, and third argument the number 5.

In the declaration, it specifies that ...

- the first argument is assigned to the first parameter, `base`
- the second argument is assigned to the second parameter, `exponent`
- the third argument is assigned to the third parameter, `thirdNum`
From Last Time

The function `performCalc` is executed with the first argument being the number 1, the second argument being the number 4, and the third argument the number 5.

When the body of the function `performCalc` is executed, the values of the local variables are used to perform the calculation.

Because `base` has the value 1, `exponent` the value 4, and `thirdNum` the value 5, the calculation is \(1 ** 4 - 5\), and because precedence of `**` is before minus, the calculation is \((1 ** 4) - 5 = 1 - 5 = -4\).
From Last Time

The function `performCalc` is executed with the first argument being the number 1, the second argument being the number 4, and third argument the number 5.

When the body of the function `performCalc` is executed, the values of the local variables are used to perform the calculation.

The function `print` “prints” to the screen -4.
When the function completes executing, **ALL** local variables (declared inside the function or declared in the function declaration) ... are **DELETED** from the computer’s memory ... those variables have only **LOCAL** scope, so they are available for use **ONLY** when that part of code is running.
From Last Time

When the function completes executing, ALL local variables (declared inside the function or declared in the function declaration) ... are DELETED from the computer’s memory ... those variables have only LOCAL scope, so they are available for use ONLY when that part of code is running

```python
def performCalc(base, exponent, thirdNum):
    print(base ** exponent - thirdNum)

performCalc(1, 4, 5)
performCalc(2, -3, 6)
performCalc(3, 7, 8)
performCalc(4, 3, 2)
performCalc(8, 2, 3)
```
From Last Time

The function `performCalc` is executed with the first argument being the number 2, the second argument being the number -3, and third argument the number 6.

The function has already been declared, and the computer “executes” the function with the indicated arguments.

```python
def performCalc(base, exponent, thirdNum):
    print(base ** exponent - thirdNum)

performCalc(1, 4, 5)
performCalc(2, -3, 6)
performCalc(3, 7, 8)
performCalc(4, 3, 2)
performCalc(8, 2, 3)
```
From Last Time

The function `performCalc` is executed with the first argument being the number 2, the second argument being the number -3, and third argument the number 6.

The function has already been declared, and the computer “executes” the function with the indicated arguments.
From Last Time

The function `performCalc` is executed with the first argument being the number 2, the second argument being the number -3, and third argument the number 6.

When the body of the function is being executed, it is using the local variables `base`, `exponent`, and `thirdNum`, thus the calculation is:

\[ 2^{-3} - 6 = (2^{-3}) - 6 = \frac{1}{8} - 6 = 0.125 - 6 = -5.875 \]
The function `performCalc` is executed with the first argument being the number 2, the second argument being the number -3, and third argument the number 6

The function `print` thus “prints” the digits -5.875 to the screen
From Last Time

```python
def performCalc(base, exponent, thirdNum):
    print(base ** exponent - thirdNum)

performCalc(1, 4, 5)
performCalc(2, -3, 6)
performCalc(3, 7, 8)
performCalc(4, 3, 2)
performCalc(8, 2, 3)
```

As before, once the function `performCalc` completes executing its code, all local variables of that function are deleted, and the computer continues to execute the reminder of the code ... eventually printing:

-4
-5.875
2179
62
61
Review – Functions that return values

```python
def performCalculation(firstNumber, secondNumber):
    if (firstNumber == 45):
        return 1 * secondNumber
    else:
        return 2 * secondNumber

outputValue = performCalculation(45, 2)
outputValue = performCalculation(34, outputValue)
print(outputValue)
```

Q: what does this python program print to the screen?

A. 0   F. 8
B. 2   G. 10
C. 4   H. 64
D. 34  I. 79
E. 45  J. 90
This function has 2 parameters, and returns something (in this case a number)
This function has 2 parameters, and returns something (in this case a number)

Recall the assignment operator ... first EVERYTHING on the right of the = is evaluated, and the “output” is used to set the value of the variable on the left side of the = sign
Review – Functions that return values

The first parameter is assigned 45, and the second parameter is assigned the value 2 ... these two, `firstNumber` and `secondNumber`, are the local variables

Whether you visualize this using the black box analogy, or by looking at the code, the behavior is the same ... the function `performCalculation` is “receiving” as input 2 values
Review – Functions that return values

def performCalculation(firstNumber, secondNumber):
    if (firstNumber == 45):
        return 1 * secondNumber
    else:
        return 2 * secondNumber

outputValue = performCalculation(45, 2)
outputValue = performCalculation(34, outputValue)
print(outputValue)

The first parameter is assigned 45, and the second parameter is assigned the value 2 ... these two, firstNumber and secondNumber, are the local variables

You can name your local variables whatever you want (the first parameter can be called aNum instead of firstNumber), but the ORDER is important. performCalculation(2, 45) is NOT the same as performCalculation(45, 2)
In the body of the function, the Boolean expression is evaluated, which evaluates to True, because \texttt{firstNumber} is equal to the number 45.
The code block for the if (when True) is evaluated, which contains a `return` statement, which first calculates `1 *` the value stored in the local variable `secondNumber`, thus `1 * 2`
Review – Functions that return values

As soon as a `return` statement is executed, that function finishes and produces the specified output … EVEN if there is more code. Think of a `return` as “leave now” … therefore this function returns (outputs) 2

```
def performCalculation(firstNumber, secondNumber):
    if (firstNumber == 45):
        return 1 * secondNumber
    else:
        return 2 * secondNumber

outputValue = performCalculation(45, 2)
outputValue = performCalculation(34, outputValue)
print(outputValue)
```
Now that the right hand side of the assignment statement has finished executing, whatever variable exists on the left hand side of the `=` is updated to be the value of the output of the function.
On the next line of code, the second argument that is provided as input to the function `performCalculation` is the value of the variable `outputValue`
Review – Functions that return values

```python
def performCalculation(firstNumber, secondNumber):
    if (firstNumber == 45):
        return 1 * secondNumber
    else:
        return 2 * secondNumber

outputValue = performCalculation(45, 2)
outputValue = performCalculation(34, outputValue)
print(outputValue)
```

Does the value of `firstNumber` equal to 45?
Review – Functions that return values

No. Therefore the else clause is executed, which contains a return statement, thus the function performCalculation returns the result of the computation $2 \times \text{secondNumber}$, which is $2 \times 2 = 4$
Review – Functions that return values

```python
def performCalculation(firstNumber, secondNumber):
    if (firstNumber == 45):
        return 1 * secondNumber
    else:
        return 2 * secondNumber

outputValue = performCalculation(45, 2)
outputValue = performCalculation(34, outputValue)
print(outputValue)
```

The value of the variable `outputValue` is updated.
Review – Functions that return values

```python
def performCalculation(firstNumber, secondNumber):
    if (firstNumber == 45):
        return 1 * secondNumber
    else:
        return 2 * secondNumber

outputValue = performCalculation(45, 2)
outputValue = performCalculation(34, outputValue)
print(outputValue)
```

Q: what does this python program print to the screen?

A. 0 F. 8
B. 2 G. 10
C. 4 H. 64
D. 34 I. 79
E. 45 J. 90

The correct answer is C. 4
Review – Function composition

```python
def functionY(aNumber, anotherNumber):
    aVariable = aNumber
    for x in range(0, 3):
        aVariable = aVariable + 1
    return anotherNumber ** 2

def functionZ(inputNum):
    return functionY(inputNum, 7)

def main():
    print(functionZ(3))

main()
```

Q: what does this python program print to the screen?

A. 0  B. 3  C. 4  D. 7  E. 8  F. 10  G. 24  H. 49
Review – Function composition

```python
def functionY(aNumber, anotherNumber):
    aVariable = aNumber
    for x in range(0, 3):
        aVariable = aVariable + 1
    return anotherNumber ** 2

def functionZ(inputNum):
    return functionY(inputNum, 7)

def main():
    print(functionZ(3))

main()
```

Q: what does this python program print to the screen?

A. 0  B. 3  C. 4  D. 7  E. 8  F. 10  G. 24  H. 49

On the board walk-through of the code
Review – Function composition

```python
def functionY(aNumber, anotherNumber):
    aVariable = aNumber
    for x in range(0, 3):
        aVariable = aVariable + 1
    return anotherNumber ** 2

def functionZ(inputNum):
    return functionY(inputNum, 7)

def main():
    print(functionZ(3))
main()
```

Q: what does this python program print to the screen?

A. 0
B. 3
C. 4
D. 7  ✔️
E. 8
F. 10
G. 24
H. 49

On the board walk-through of the code
Review – Function composition

```
def functionY(aNumber, anotherNumber):
    aVariable = aNumber
    for x in range(0, 3):
        aVariable = aVariable + 1
    return anotherNumber ** 2

def functionZ(inputNum):
    return functionY(inputNum, 7)

def main():
    print(functionZ(3))
```

A follow up question ...
If the code is changed so that the main function is not invoked, what is the output of this program?

A. 0  E. 8  
B. 3  F. 10  
C. 4  G. 24  
D. 7  H. 49
Today

Strings
Strings

Motivation

We are accustomed to performing operations on numbers, for example adding, multiplying, dividing, etc.

Unless you’ve worked with “string” and/or textual data before, it might not be clear why do we want (need?) the ability to manipulate strings.
Strings

Motivation

We are accustomed to performing operations on numbers, for example adding, multiplying, dividing, etc.

Unless you’ve worked with “string” and/or textual data before, it might not be clear why do we want (need?) the ability to manipulate strings

Assume you have the following program:

```python
usersAddress = input("Please input your house number and street name ")
```
Strings

Motivation

We are accustomed to performing operations on numbers, for example adding, multiplying, dividing, etc.

Unless you’ve worked with “string” and/or textual data before, it might not be clear why do we want (need?) the ability to manipulate strings

Assume you have the following program:

```python
usersAddress = input("Please input your house number and street name ")
```

When run, and the user inputs “6249 Columbus”, what is saved into the variable `usersAddress`?
Strings

Motivation

We are accustomed to performing operations on numbers, for example adding, multiplying, dividing, etc.

Unless you’ve worked with “string” and/or textual data before, it might not be clear why do we want (need?) the ability to manipulate strings.

Assume you have the following program:

```python
usersAddress = input("Please input your house number and street name ")
```

When run, and the user inputs “6249 Columbus”, what is saved into the variable `usersAddress`?
Motivation

We are accustomed to performing operations on numbers, for example adding, multiplying, dividing, etc.

Unless you’ve worked with “string” and/or textual data before, it might not be clear why do we want (need?) the ability to manipulate strings

Assume you have the following program:

```python
usersAddress = input("Please input your house number and street name ")
```

What if want to write a program that would then output

“Your neighbor’s house number is 6251”

Q: What is required?
Strings

The problem: We cannot “add” 2 to 6249 Columbus because that is a string, and we cannot convert 6249 Columbus to a number because letters do not convert to numbers.

Therefore, we need some sort of functionality to manipulate String objects.

```python
usersAddress = input("Please input your house number and street name ")
```

What if we want to write a program that then would then output “Your neighbor’s house number is 6251”
What does the following program print to the screen?

```python
aWord = "happy"
print(aWord * 3)
```

A. Nothing – Syntax error
B. happy3
C. aWord3
D. aWordaWordaWord
E. happyhappyhappy
F. aHappyWord3
Strings

What does the following program print to the screen?

```python
aWord = "happy"
print(aWord * 3)
```

A. Nothing – Syntax error
B. happy3
C. aWord3
D. aWord aWord aWord
E. happy happy happy
F. aHappyWord3

Q: What other combinations of operands are allowed when using the * operator?

Q: Can the * operator be used when the left hand operand is an integer and the right hand operand is a string?

(live demo)
We’ve seen `int`, `float`, `bool`, `str`, as well as `lists`. All of these are data types that permit us (programmers) to hold on to and manipulate data of different types.

Q: How are `int`, `float`, `bool` different from `str` and `lists`?
We’ve seen `int`, `float`, `bool`, `str`, as well as `lists`. All of these are data types that permit us (programmers) to hold on to and manipulate data of different types.

These two are collection data types because they are made up of “smaller” pieces.

- A string is made up of characters
- A list is made up of 1 or more items
**Empty string**: a sequence of zero characters. For example `someString = ""`.

This is an important concept because a function might return an empty string. For example consider the hypothetical function `extractAllNumbers`, which returns a string:

```
"62 Columbus Ave"
```

```
extractAllNumbers
```

```
"62"
```
Empty string: a sequence of zero characters. For example `someString = ""`

This is an important concept because a function might return an empty string. For example consider the hypothetical function `extractAllNumbers`, which returns a string:

- "62 Columbus Ave" -> `extractAllNumbers` -> "62"
- "sixty-two Columbus Ave" -> `extractAllNumbers` -> "62"

Q: What should the function `extractAllNumbers` return in this case?
Empty string: a sequence of zero characters. For example `someString = ""

This is an important concept because a function might return an empty string. For example consider the hypothetical function `extractAllNumbers`, which returns a string:

- "62 Columbus Ave" → `extractAllNumbers` → "62"
- "sixty-two Columbus Ave" → `extractAllNumbers` → ""

The Empty string
We’ve seen that * works with strings, but in all cases? And what about other “math” operators where one of the operands is a string?

**Assuming**: `aSillyString = “I am a silly string”`

**Q**: Which of the following is/are allowed?

```python
print(3.0 * aSillyString)
aSillyString - 1
aSillyString / 34
aSillyString * "3.0"
print(aSillyString * int("3.0"))
aSillyString * int("3")
```
Operations on Strings

We’ve seen that * works with strings, but in all cases? And what about other “math” operators where one of the operands is a string?

**Assuming**: `aSillyString = “I am a silly string”`

_Q: Which of the following is/are allowed?_

Only this one, because `int(“3”)` converts the string “3” into an integer, and we’ve already seen that a String can be “multiplied.” All of the others are not allowed.

- `print(3.0 * aSillyString)`
- `aSillyString - 1`
- `aSillyString / 34`
- `aSillyString * “3.0”`
- `print(aSillyString * int(“3.0”))`
- `aSillyString * int(“3”)`

This means that if you use the * operator on a String, the non-string operand must be an integer.
Operations on Strings

The + operator also works with strings. When the operands are strings, then the + is the **concatenation** operator.

```python
firstWord = "banana"
secondWord = "who said that"
print(firstWord + secondWord)
```

**What does the above code print to the screen?**
The $+$ operator also works with strings. When the operands are strings, then the $+$ is the **concatenation** operator.

```python
firstWord = "banana"
secondWord = "who said that"
print(firstWord + secondWord)
```

What does the above code print to the screen?

```
bananawho said that
```
Operations on Strings

Multiple concatenations are allowed ...

```python
firstWord = "banana"
secondWord = "who said that"
print (firstWord + secondWord + firstWord)
```

Q: What is printed to the screen?
Operations on Strings

Multiple concatenations are allowed ...

```python
firstWord = "banana"
secondWord = "who said that"
print (firstWord + secondWord + firstWord)
```

Q: What is printed to the screen?

```
bananawho said thatbanana
```
Operations on Strings

What is the output of the following program?

```python
aDogName = "Fido"
print((aDogName + "growls") * 2)
```

A. Fido  
B. Fidogrowls  
C. Fidogrowls2  
D. FidogrowlsFidogrowls  
E. Fido growls Fido growls  
F. None of the above
Operations on Strings

What is the output of the following program?

```python
aDogName = "Fido"
print((aDogName + "growls") * 2)
```

This concatenates the string “Fido” with “growls”

A. Fido  
B. Fidogrowls  
C. Fidogrowls2  
D. FidogrowlsFidogrowls  
E. Fido growls Fido growls  
F. None of the above
Operations on Strings

What is the output of the following program?

```python
aDogName = "Fido"
print((aDogName + "growls") * 2)
```

Which then is printed out twice

A. Fido  
B. Fidogrowls  
C. Fidogrowls2  
D. FidogrowlsFidogrowls  
E. Fido growls Fido growls  
F. None of the above
Operations on Strings

What is the output of the following program?

```
aDogName = "Fido"
print((aDogName + "growls") * 2)
```

Which then is printed out twice

A. Fido
B. Fidogrowls
C. Fidogrowls2
D. FidogrowlsFidogrowls
E. Fido growls Fido growls
F. None of the above

Important: Notice that the append operator does NOT add spaces between strings.
Operations on Strings

There are other functions that can be used to manipulate strings ... but first a closer look at strings ...

Because strings are made up of “smaller” strings (characters), you should think of each character in a string occupying a unique “position” in the string.

\[ \text{aWord} = \text{“SuMmERs”} \]

We need a mechanism to uniquely identify each character in a string ... we want to give each character a unique ID
There are other functions that can be used to manipulate strings ... but first a closer look at strings ...

Because strings are made up of “smaller” strings (characters), you should think of each character in a string occupying a unique “position” in the string.

\[ \text{aWord} = \text{"SuMmERs"} \]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>u</td>
<td>M</td>
<td>m</td>
<td>E</td>
<td>R</td>
<td>s</td>
</tr>
<tr>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>
Operations on Strings

There are other functions that can be used to manipulate strings ... but first a closer look at strings ...

Because strings are made up of “smaller” strings (characters), you should think of each character in a string occupying a unique “position” in the string.

\[
aWord = \text{“SuMmERs”}
\]

The character S is at position 0 of the String, and position -7
There are other functions that can be used to manipulate strings ... but first a closer look at strings ...

Because strings are made up of “smaller” strings (characters), you should think of each character in a string occupying a unique “position” in the string.

\[
aWord = \text{“SuMmERs”}
\]

The character \(S\) is at position 0 of the string, and position -7.

Important: All strings in Python, when referring to the index when read from left to right, “begin” with 0. But all strings, when referring to the index when read from right to left, “begin” with -1.
With that information (indices) we can now “extract” information from a string. You use the [] brackets to specify the position of a character.

```python
anotherSillyString = "Fred"
print(anotherSillyString[0])
print(anotherSillyString[-2])
```

Q: What is the output of the above program?
With that information (indices) we can now “extract” information from a string. You use the [] brackets to specify the position of a character.

```
anotherSillyString = "Fred"
print(anotherSillyString[0])
print(anotherSillyString[-2])
```

The [0] means “go to the string with name anotherSillyString, and what is the character (string) at the position 0?”
Operations on Strings

With that information (indices) we can now “extract” information from a string. You use the [ ] brackets to specify the position of a character.

```python
anotherSillyString = "Fred"
print(anotherSillyString[0])
print(anotherSillyString[-2])
```

The [-2] means “go to the string with name anotherSillyString, and what is the character (string) at the position -2?”

Q: What is the output of this program?
Operations on Strings

With that information (indices) we can now “extract” information from a string. You use the [] brackets to specify the position of a character.

```python
anotherSillyString = "Fred"
print(anotherSillyString[0])
print(anotherSillyString[-2])
```

The [-2] means “go to the string with name anotherSillyString, and what is the character (string) at the position -2?”

F
e
Operations on Strings

With that information (indices) we can now “extract” information from a string. You use the [] brackets to specify the position of a character.

```
anotherSillyString = "Fred"
print(anotherSillyString[0])
print(anotherSillyString[-2])
print(anotherSillyString[-5])
```

Q: What is the output of this last statement?
With that information (indices) we can now “extract” information from a string. You use the `[]` brackets to specify the position of a character.

```python
anotherSillyString = "Fred"
print(anotherSillyString[0])
print(anotherSillyString[-2])
print(anotherSillyString[-5])
```

Q: What is the output of this last statement?

An index out of range error (`IndexError`)
Operations on Strings

With knowledge of indices, you can “extract” information from a string ... this is the problem that we started out with.

Task: Assume the below has been declared:

```
anAddress = "1800 Pennsylvania Ave."
```

Write code that will print to the screen:

- President’s home address is 1800
- President’s street is Pennsylvania

You cannot use the string literal “President’s home address is 1800” nor “President’s street is Pennsylvania” (there is more than one solution)
Operations on Strings

```python
anAddress = "1800 Pennsylvania Ave."

houseNum = ""
for i in range (0,4):
    houseNum = houseNum + anAddress[i]

street = ""
for i in range(5,17):
    street = street + anAddress[i]

print("President's home address is " + houseNum)
print("President's street is " + street)
```
Take home exercise

Write the function `extractAllNumbers` that

• Has a single parameter (which you can assume is of type string)
• It detects all characters in the input argument that are digits, from left to right
• Returns a String that contains all of the digits in the input string

For example, the below program

```python
aString = "There are 72 months in 6 years"
print(extractAllNumbers(aString))
```

Would print 726
Next up

More string functionality