Information

Monday, June 3, 2024 9:57 AM

Grading:

- Problem sets: 70%
  - 4 total (Chapters 3.2, 3.3, 3.4, 3.5)
  - Unlimited attempts
- Test: 30%
  - Limited attempts
- Practice tests will be given in preparation for the test, which will be 2 hours timed and include multiple choice, fill in the blank, and live coding problems.

3.1 Control Structures

Wednesday, June 5, 2024 9:48 AM

Control Structures?

Control Structures: statements that control the flow of execution of the program; lines of code that control when other lines run. Generally can be broken up into four types: conditionals, loops, functions, and exception handling.

Conditional Statements: programming statements that control what code is executed based on certain conditions; usually of the form "if, "else if", and "else". Conditionals have to be properly formatted with the correct indentations.

Loop: a control structure that executes a segment of code multiple times.

Function: a segment of code that performs a specific task, sometimes taking some input and return some output.

Error Handling:

- Exception: an error that a program might want to anticipate and catch instead of outright avoiding.
- Exception Handling: a control structure that catches certain anticipated errors and reacts to them accordingly.

Scope: the portion of a program's execution during which a variable can be seen and accessed. The way scope works is generally very different between scripting and compiled languages. Scope is usually defined by control structures; variables created within a conditional statement cannot be seen outside of that conditional, for example. For Python in general, every variable or function created has a line on which it was made. If that line has already run, then the scope of that variable or function has begun. If it hasn't, then the variable's scope hasn't.

3.2 Conditionals

Wednesday, June 5, 2024 9:48 AM

Conditionals

Conditional Statements: programming statements that control what code is executed based on certain conditions, usually in the form of "if", "else if", and "else". The most fundamental form of this statement is "if-then", which runs a block of code only if certain conditions are met. With an "if-then-else" structure, an alternative block of code is run if the conditions aren't met. With an "if-then-else-if" structure, multiple blocks of code can be run based off various conditions being met. If multiple "else-ifs" are used, an else if only executes if no previous conditions were met.

Conditionals can use logical and mathematical operators.

Conditionals in Python

If-then:

```
# if-then
todaysWeather = "raining"
#Checks if todaysWeather equals "raining"
if todaysWeather == "raining":
    #Prints "raincoat" if so
    print("raincoat")
    #Prints "rainboots" if so
    print("rainboots")
#Prints "Done!" when complete
print("Done!")
```

If-then-else:

```
todaysWeather = "cold"
#Checks if todaysWeather equals "raining"
if todaysWeather == "raining":
    print("raincoat")
    print("rainboots")
#If todaysWeather didn't equal "raining",
#do the following
else:
    print("t-shirt")
    print("shorts")
print("Done!")
```

If-then-else-if-else:

```
#Creates todaysWeather and sets it equal to "cold"
todaysWeather = "cold"
#Checks if todaysWeather equals "raining"
if todaysWeather == "raining":
    print("raincoat")
    print("rainboots")
#Otherwise, checks if todaysWeather equals "cold"
elif todaysWeather == "cold":
```

```
    print("long-sleeved shirt")
    print("scarf")
#If todaysWeather didn't equal "raining", do the following
else:
    print("t-shirt")
    print("shorts")
print("Done!")
```

```
item = "quesadilla"
meat = "steak"
queso = False
guacamole = False
double_meat = False
base_price = 4.5
if item == "quesadilla":
    base_price = 4.0
elif item == "burrito":
    base_price = 5.0

if meat in ("steak", "pork"):
    base_price += 0.50
if meat in ("steak", "pork") and double_meat:
    base_price += 1.50
if meat not in("steak", "pork") and double_meat:
    base_price += 1.0
if guacamole:
    base_price += 1.0

if item != "nachos" and queso:
    base_price += 1.0

print(base_price)
```

Nested Conditionals

Nested Conditional: a conditional statement that is itself controlled by another conditional statement. It is an if-then statement within another if-then statement.

Nested conditonal:

```
x = 3
if x > 5:
    print("x is greater than 5.")
    if x > 0:
        print("x is greater than 0.")
# nothing prints because the nested conditional only runs if the first conditional is
true.
```

Non-nested conditional:

```
if x > 5:
    print("x is greater than 5.")
if x > 0:
    print("x is greater than 0.")
```

```
entered = "abc123"
password = "abc123"
tries = 3

if tries > 3:
    print("Tries exceeded.")

else:
    if entered == password:
        print("Login successful.")
    else:
        print("Incorrect password.")
```

3.3 Loops

Wednesday, June 5, 2024 9:48 AM

What is a loop?

Loop: a programming control structure that executes a segment of code multiple times. Two major types of loops are For Loops and While Loops.

For Loop: a loop control structure that runs a block of code for a predetermined number of times (or iterations), using range().

While Loop: a loop control structure that runs a block of code until a certain logical expression is satisified.

For-Each Loop: a loop control structure that runs a block of code a predetermined number of times, where the number fo times comes from the length of some list and the items in the list are automatically loaded into a variable for usage in the block of code.

There is nothing you can do with a For Loop that you cannot do with a For-Each loop or vice versa.

For Loops in Python

Loop Control Variable: a variable whose value is the number of times a loop has run, used to check if the loop should continue running or if it has run as many times as it is supposed to.

range(begin, end): takes as input two variables, a starting and ending number, and provides the list of numbers for a For Loop to iterate over. Note that the loop stops before the end value; it is not inclusive of `end`. Other variations of this function include:

- range(begin, end, step size), where step size is how much the count up amount is (default = 1). A step size of 2 would count 2, 4, 6, etc.
- range(begin, end, -(step size)). If the beginning value is larger than the ending value and the step size is negative, the function counts down instead of up.
- range(number): if no beginning value is supplies, it is assumed to start at zero.

```
# for loop with range
mystery_int = 6
factorial = 1
for i in range(1, mystery_int+1):
    factorial *= i # multiply factorial by i each loop
print(factorial)
```

```
# for-each loop
listOfNumbers = [91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
sum = 0
#Runs this loop once for each item, assigning the current
#item to the variable 'currentNumber'
for currentNumber in listOfNumbers:
    sum += currentNumber
#Divides sum by the number of items in the list to get the average
print(sum / len(listOfNumbers))
```

```
# for-each with if statement
myString = "There are seven words in this string."
numSpaces = 0
#Runs this loop for each character in the string
for currentCharacter in myString:
    #Checks if the character is a space
    if currentCharacter == " ":
        numSpaces += 1
print("There are " + str(numSpaces + 1) + " words in the string.")
```

While Loops in Python

Any For Loop can be written as a While Loop. With While Loops, you have to take care that you don't get caught in an infinite loop.

```
# while loop guessing game
import random
#Get a random number from 1 to 100
hiddenNumber = random.randint(1, 100)
#Create userGuess and give is a value that can't be correct
userGuess = 0
#Repeat until the guess is correct
while not userGuess == hiddenNumber:
    #Get the user's next guess as an integer
    userGuess = int(input("Guess a number: "))
    #Check if the guess is too high
    if userGuess > hiddenNumber:
        print("Too high!")
    #Check if the guess is too low
    elif userGuess < hiddenNumber:
        print("Too low!")
    #The guess must be right!
    else:
        print("That's right!")
```

Nested Loops

```
beats_per_measure = 4
measures = 5
for num in range(measures):
    for beats in range(1, beats_per_measure+1):
        print(beats)
```

Loop Keywords and Scope

Continue: skip the rest of the current iteration of the loop and continue with the next iteration of the loop (if there is one)

Break: skip the rest of the current iteration of the loop and break out of the loop altogether, skipping later iterations

Pass: designate an "empty" body for a control structure; a blank indented line.

```
# continue
#Runs this loop 20 times
for i in range(1, 21):
    #Checks if i is even
    if i % 2 == 0:
        #Skips the rest of the code block if so
        continue
    #Prints that i is odd
    print(i, "is odd.")
print("Done!")
```

```
# break
#Runs this loop 20 times
for i in range(1, 21):
    #Checks if i is even
    if i % 2 == 0:
        #Skips the rest of the loop if so
```

```
            break
        #Prints that i is odd
        print(i, "is odd.")
print("Done!")
```

3.4 Functions

Wednesday, June 5, 2024 9:48 AM

Functions

Function: a segment of code that performs a specific task, sometimes taking an input and sometimes returning an output. Functions have two major parts: the definition and the call.

Function Definition: made up of the header and body, this creates the function - including its name, parameters, and code - to be used by other portions of a program.

Parameter: a variable for which a function expects to receive a value when called, whose scope is the function's own execution.

Function Header: the name and list of parameters a function expects, provided as a reference to the rest of the program to use when calling the function.

Function Body: the code that the function runs when called.

Return statement: the line of code that defines what output willb e sent back at the end of a function.

Function Call: a place where the function is actually used in some code.

Arguments: values passed into parameters during a function call. These are the values assigned to the function's dedicated parameters.

Keyword Parameters: a special kind of optional parameter to which the program may choose to assign an argument during a function call or they may ignore it. Typically, they have a default valuethat is used if it is not overridden by the function call.

Common Function Errors

Parameter Mismatch: giving a function more or fewer parameters than it expected or is defined to have.

Scope Error: using a variable in a function that was created outside of the function or using a variable outside of a function that was created in the function. Examples include the sep and end keyword parameters from the print() function.

Examples

```
#1
from datetime import date

#Write your function here!
def get_todays_date():
    today = date.today()
    date_format = (str(today.year) +"/"+ str(today.month) +"/"+ str(today.day))
    return date_format

print(get_todays_date())

#2
def reverse(a_string):
    #You may add code before the following three lines.
    rev = ""

    #Don't change these three lines.
    for i in range(len(a_string)-1, -1, -1):
        rev = rev + a_string[i]
    return rev

#You may change or add to the following lines.
rev = reverse("This string should be reversed!")
print(rev)

#3
```

```
def snowed_in(temperature, weather, is_celsius = True):
    #keyword parameter - is_celsius
    #Note that we could accomplish this entire function
    #in only one line. Here's the line:

    return weather == "snowy" or temperature < 0 or (temperature < 32 and not is_celsius)
print(snowed_in(15, "sunny")) #Should print False
print(snowed_in(15, "sunny", is_celsius = False)) #Should print True
print(snowed_in(15, "snowy", is_celsius = True)) #Should print True
```

3.5 Error Handling

Wednesday, June 5, 2024 9:48 AM

Exception Handling

Some languages use "error" and "exception" interchangeably, but other languages treat them separately.

Catching Errors: using error handling to prevent a program from crashing when an error is encountered. Benefits of error catching include:

- Improved program design thought process
- Organized code
- Preventing unanticipated errors from crashing the program

Uncaught Errors: an error that is not handled by error handling code and thus usually forces the program to crash.

Try and Except

Try: this statement marks a block of code to attempt, but in which we anticipate an error may arise. This is a control structure t hat sets aside a block of code in which an error might occur so that the computer will look for error handling capabilities.

Catch: this statement names the error to anticipate and marks a block of code to run if the anticipated error arises. Python refers to this as the Except block. This is a control structure that designates what error it anticipates in a try block and provides the code to execute if that error arises.

Finally: this statement marks a block of code to run after the Try and Catch blocks, no matter what. This is a control structure that designates some code to run after a try and catch structure, regardless of whether or not an error arose.

In Python, there is also an Else block: it runs some code if no errors arose in the Try block.

```
# run below code until an error occurs.
try:
    div = 10/mystery_value
    print(div)

#If an error occurs, check if it's a ZeroDivisionError
except ZeroDivisionError as error:
    print(error)

#If an error occurs, check if it's a NameError
except NameError as error:
    print(error)

#If an error occurs, check if it's a ValueError
except ValueError as error:
    print(error)

#If an error occurs, check if it's a TypeError
except TypeError as error:
    print(error)

# if any other error occurs, jump to this block
except:
    print("Not possible")
```

Note: if you have multiple errors, only the first one that the try block hits will occur; only the first one will go to its respe ctive except block.

Else and Finally

Else: this statement executes code when there is no error.

Finally: this statement executes code, regardless of the result of the try-except blocks.

These blocks can also be nested.

Note: using a Return statement ends the function

```
try:
    #Open NumberFile.txt in read-only mode
    input_file = open("NumberFile.txt", mode = "r")
    try:
        #For each line in the file
        for line in input_file:
            #Print the line
            print(int(line))
    #Catch a ValueError
    except ValueError as error:
        print("A value error occurred!")
    else:
```

```
        print("No errors occurred converting the file!")
    finally:
        #Close the file
        input_file.close()
#Catch an IOError
except IOError as error:
    print("An error occurred reading the file!")


# incorrect
for num in range(-3, 3):
    try:
        print(5 / num)
    except:
        pass
# correct
try:
    for num in range(-3, 3):
        print(5 / num)
except:
    pass
```

Conclusion

Wednesday, June 5, 2024 9:49 AM