Intro to Scripting Python

Module 3

Show answer to homework:

Restaurant.py

Constants:

Often in a program, you will have a need for a number that doesn't change. For example, in the homework, the cost of a hamburger was 3.00. While the program is running, that value will never change. So, you could write a line of code where you calculate the cost of hamburgers my multiplying the number of hamburgers by 3.00. However, the cost of a milkshake was also 3.00.

Now imagine you that as the restaurant owner, you decide that you need to raise the price of your hamburger from 3.00 to 3.25. To update your program, you have to find every occurrence of 3.00 in your program, and decide on a line by line basis if the 3.00 represents a hamburger or a milk shake, and only change the appropriate one(s). While this is very simple in our little restaurant program. This type of change can take a great deal of time and be very error-prone in a large program.

In Python, we create a variable for a number like this, just so it can be referred to by name throughout the program.

+++++
DEF: <u>Constant</u> – a variable whose value does not change throughout a program

Python Convention – a constant uses a naming convention of all upper case letters separated by underscores. This serves as a signal to yourself and other programmers that its value should never be reassigned.

Examples:

```
COST_PER_HAMBURGER = 3.00
COST_PER_HOT_DOG = 2.00
COST_PER_MILK_SHAKE = 3.00
```

Then use COST_PER_HAMBURGER, COST_PER_HOT_DOG and COST_PER_MILK_SHAKE in the program instead of the prices. That way, if the cost of either or both changes, then you only have to make the change in one place, the original assignment statement. No other changes are needed, AND your code becomes more readable.

+++++

Show: RestaurantWithConstants.py

Scope:

We’ve talked about how variables have a type (integer, float, boolean, string). But variables also have a lifetime. Let’s start with another definition:

++++
DEF: Scope – the amount of code over which a variable is active.

In Python, there are three types of scope. In this class, we will only talk about two types of scope:

++++ Global variables vs local variables.

Global variables are variables that are created at the top level of a program. They have what is called global scope. Global variables are available throughout the program, for example, before and after calls to other functions.

Global constants are good things because they clarify the code, and can be used inside any function in the whole program.

+++++ Notice that the constants: COST_PER_HAMBURGER, COST_PER_HOT_DOG, and COST_PER_MILK_SHAKE are created outside any function, and therefore have global scope. They are given values at the top level, but then they are used inside the calculateBill function.

However, while global variables (created in the main program code) *can* legally be used inside functions, I am going to teach this class in a way where I don’t want you to use global variables inside functions – only global constants. Using global variables inside functions leads to a poor coding practice called "spaghetti code", where variables are modified all over a program which makes the code very hard to understand and maintain.

Local variables are created inside a function. They only live from where they are first used in a function to the end of that function, then they disappear. The "scope" is limited to that function only.

But inside the function, we are creating and using local variables of: subTotal, tax, and total. These are local variables because they are only used inside of a function. When the function exits, these variables literally disappear. You cannot access these variables outside of the function because they do not exist any more.

All parameter variables, the ones listed in the def statement are really just local variables. parameters (or parameter variables) are given their values when the function is called, and they go away when the function is finished.

In general, when you call a function, you should pass in any values that the function needs, and the function should pass back any result that it wants to return to the caller.

You can think of it this way. When you call a function from any code, the caller's code has its set of variables which it remembers, and the called function has its set of variables which it uses.

Introduction of the “If” statement

So far, all the code we have looked at has essentially been linear. That is, it starts from the top and goes straight through. I have shown how you can make a function call, that changes the linear nature of coding by going into a function, and then returning to the caller. But so far, all the code inside functions has gone straight through from top to bottom.

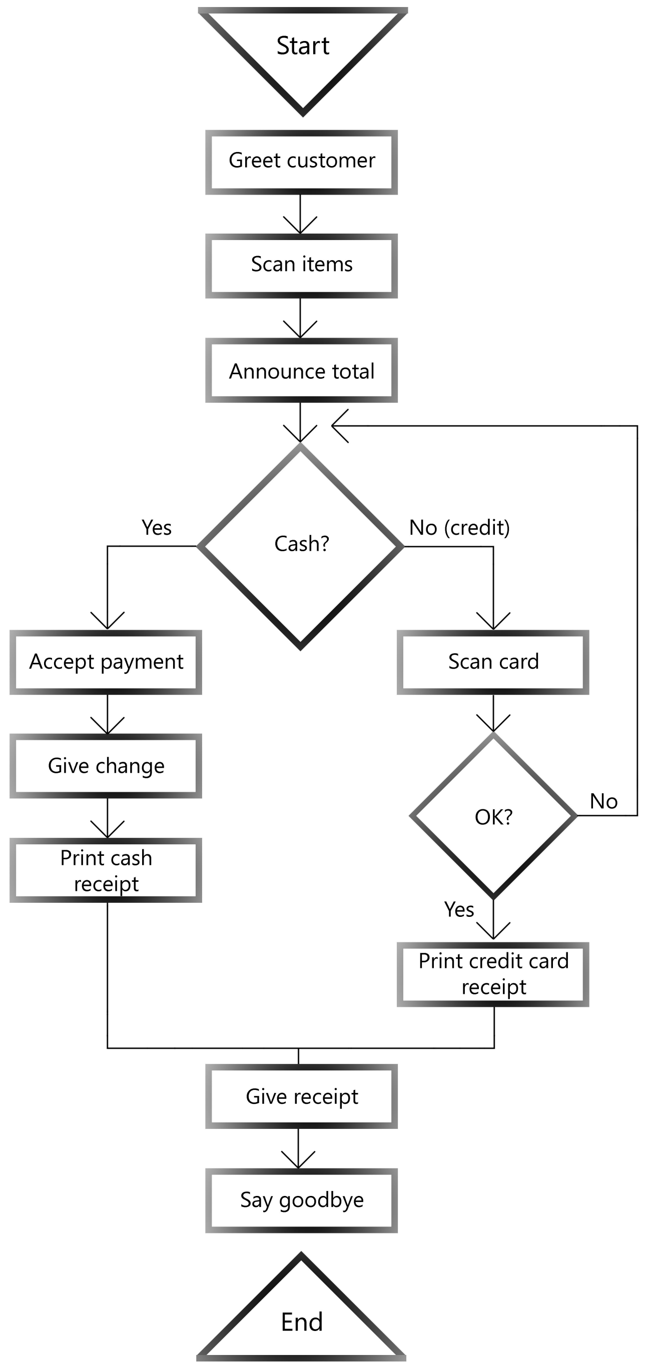
But, one of the most powerful things about code, is the ability to make a decision, and to take branches based on that decision. In the real world, this happens all the time:

If I am hungry, then I will eat
If today is XXXday, then I have programming class
If I am tired, then I will go to sleep
If I have to go far and I own a working car, then I will use my car

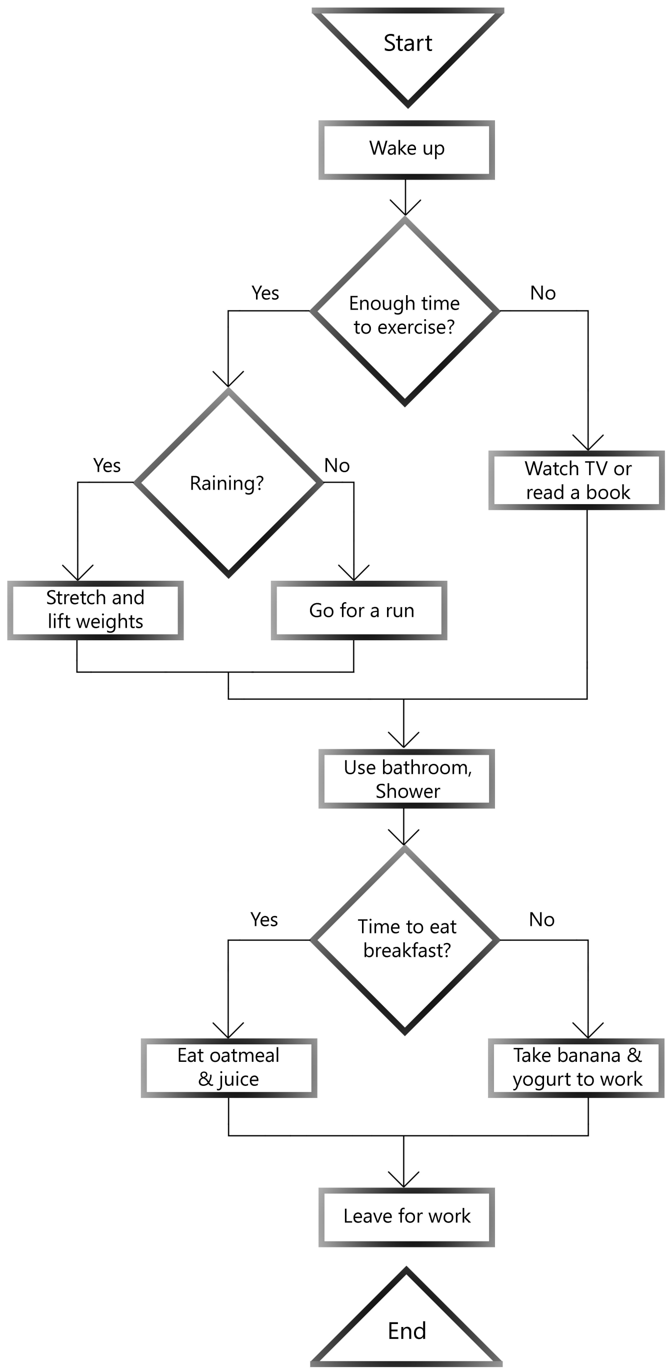
++++

Flowcharting:

Notice decision boxes (with yes/no or true/false answers)



Store check out – cash or credit.

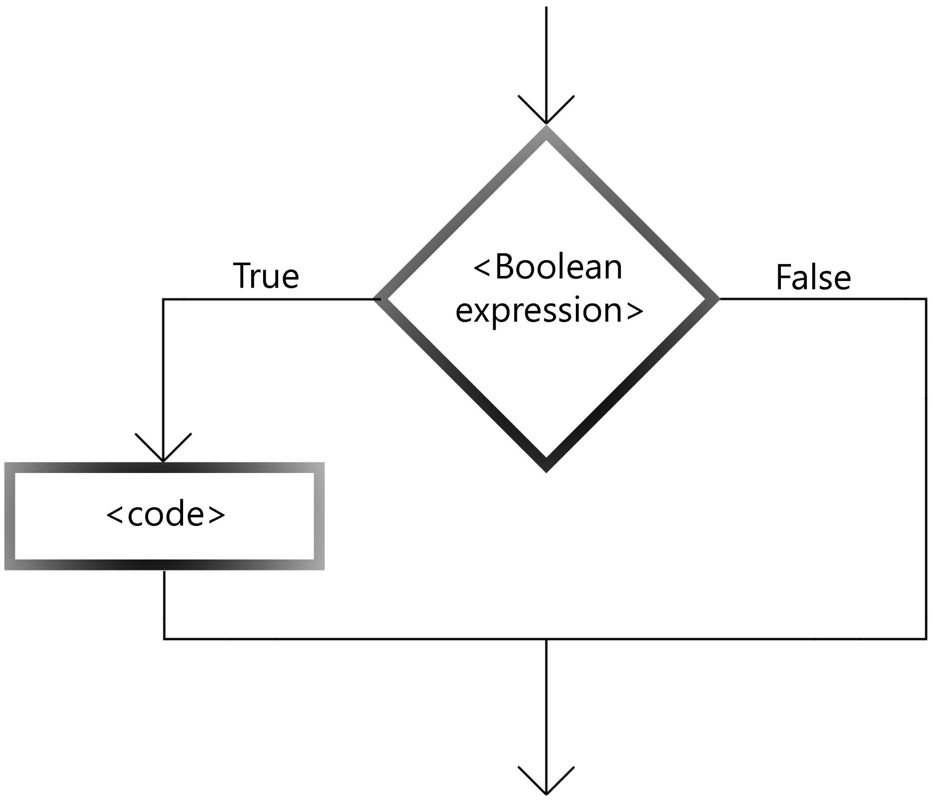


Workday morning routine.

The details of these flowcharts are not particularly important. What is important is the decision diamonds and the resulting True or False branches.

++++
if statements:

In programming, a decision box in a flowchart is implemented by an “if” statement in Python. Using an “if” statement, programmers can ask a question and only run some code if the answer is True.



++++Python syntax:

```
if <boolean expression>:
	<indentedBlockOfCode>  #any number of indented lines
```

where a <boolean expression> is an expression that evaluates to True or False

++++ Example:

```
if teachersName == ‘Irv’:
        teachingClass = True
        print(‘Pay attention to his wisdom’)
        respectPoints = respectPoints + 10


if nStudentsUnderstandingPython == 0:
        fireIrv()
        getNewTeacher()
```

++++
NOTE: The “==” is called a comparison operator. It is used in “if” statements to compare two values for equality. Remember the single equals sign “=” is called the assignment operator.

If you use a single equals sign in an if statement, you get an error message:

```
myVariable = 1
if myVariable = 1:  # This is an error, needs to be equals equals
    print('The value of myVariable is one.')
```

Notice also that all the code that is part of what to do when the boolean expression is True is indented. By convention, Python code is indented four spaces. IDLE does this for you automatically. That is, when you type an if statement and end it with a colon, IDLE automatically indents the next line four spaces and will continue to indent for you until you stop it by hitting the backspace or delete key. Four spaces is the default, but it is a setting in the preferences of IDLE.

In other languages like C, Java, and Javascript, the block of code that is executed is signified by using squiggly braces {}, and also is often indented by programmers. Python has eliminated these braces, and only uses the indentation. This makes the code more readable.

++++

Operators in if statements:

Op:	Meaning:	Example:
==	equals	if a == b:
!=	not equal	if a != b:
<	less than	if a < b:
>	greater then	if a > b:
<=	less than or equal	if a <= b:
>=	greater than or equal	if a >= b:

++++

Examples of if statements:

```
if dealersTotal < yourTotal:
    print('You win')
```

```
if age >= 18:
    allowedToVote = True
    consideredAnAdult = True


if cashInWallet > costOfGasPerGallon:
        purchaseGallonOfGas(1)
        csahInWallet = cashInWallet - costOfGasPerGallon


if userPassword != savedPassword:
     giveErrorMsgAboutPassword()
```

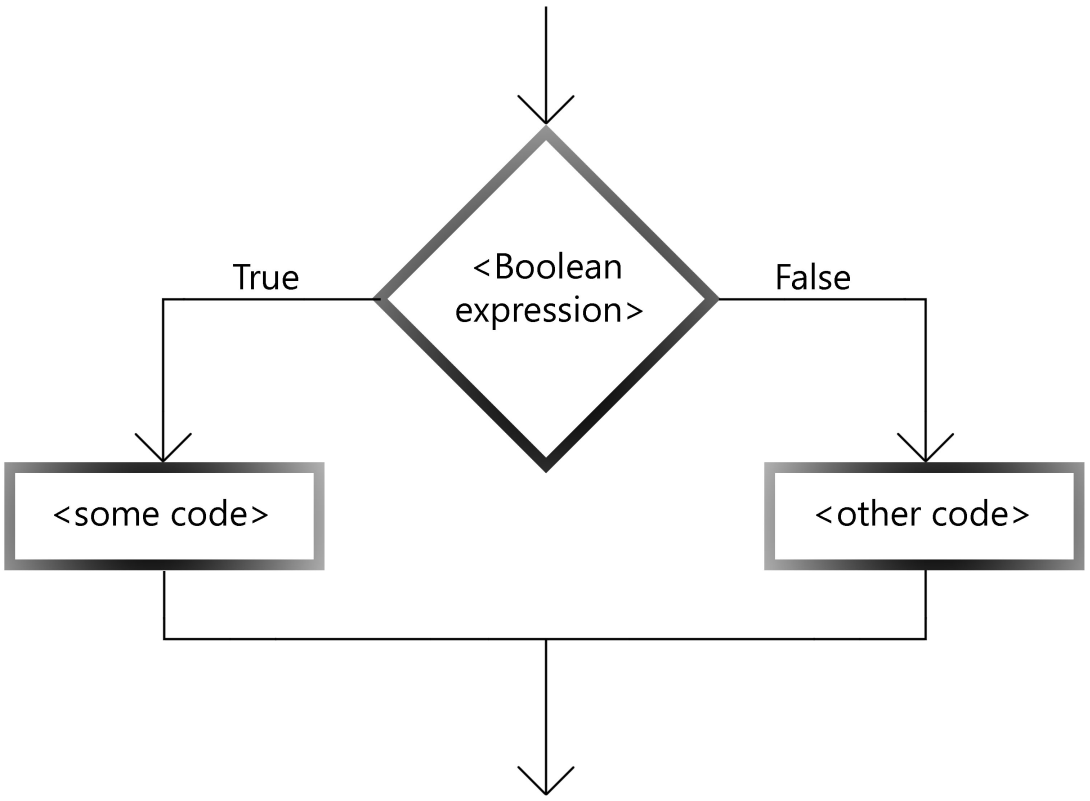
Nested if statements:

```
++++
# Purchases at a gas station
totalGasPurchase = priceOfGas * nGallons
amountLeftOver = startingAmountOfMoney –
                 totalGasPurchase

# See if we can buy a Powerball lottery ticket
if amountLeftOver > 2:
    feeling = evaluateEmotions()
    if feeling == 'lucky':
        buyPowerballTicket()
        amountLeftOver = amountLeftOver - 2
```

if/else statements:

Often, when you ask a question in an ‘if’ statement, you want to take one branch if the answer evaluates to True, and do something different if the answer is False.



Expand if statement to use else clause:

```
if <boolean expression>:
        <some indented code>
else:
        <other indented code>
```

++++

Examples of if/else statements:

```
if age < 21 then
   okToOrderBeer = False
   print('Sorry, you are too young!')
else:
   okToOrderBeer = True
   beerOrder = input('What would you like to drink?')


if gpa >= 3.5:
        applyToWorkAtGoogle()
else:
        if gpa > 3.0:
                applyToWorkAtHP()
        else:
                applyToWorkAtWalmart()


if age >= 18:
        canVote = True
else:
        canVote = False

if age >= 65:
        canCollectSocialSecurity = True
else:
        canCollectSocialSecurity = False



++++[Build the following in class]
answerToQuestion = input('What is 2 + 3? ')
answerToQuestion = int(answerToQuestion)

if answerToQuestion == 5:
        print('You got it')
        print('You are a genius')
else:
        print('Nope')
        print('You are an idiot')
```

All string comparisons are case sensitive. For example:

```
if 'ThisString' == 'thisstring':
```

will be False

++++

Using if/else inside function:

Assume you want to write a function that will put out the proper header for a letter, based on whether the person is male or female. In this function, you pass in a name and a Boolean, 'm' for male, 'f' for female:

```
def createHeader(fullName , gender):

    if gender == 'm':
        title = 'Mr. '
    else:
        title = 'Ms. '
    header = 'Dear ' + title + ' ' + fullName
    return header
```

```
print(createHeader('Joe Smith', 'm'))
print(createHeader('Susan Jones', 'f'))
print(createHeader('Henry Jones', 'm'))
```

What if we didn't know the gender of each person. Let's say we pass in a gender of 'm', 'f', or '?'

+++

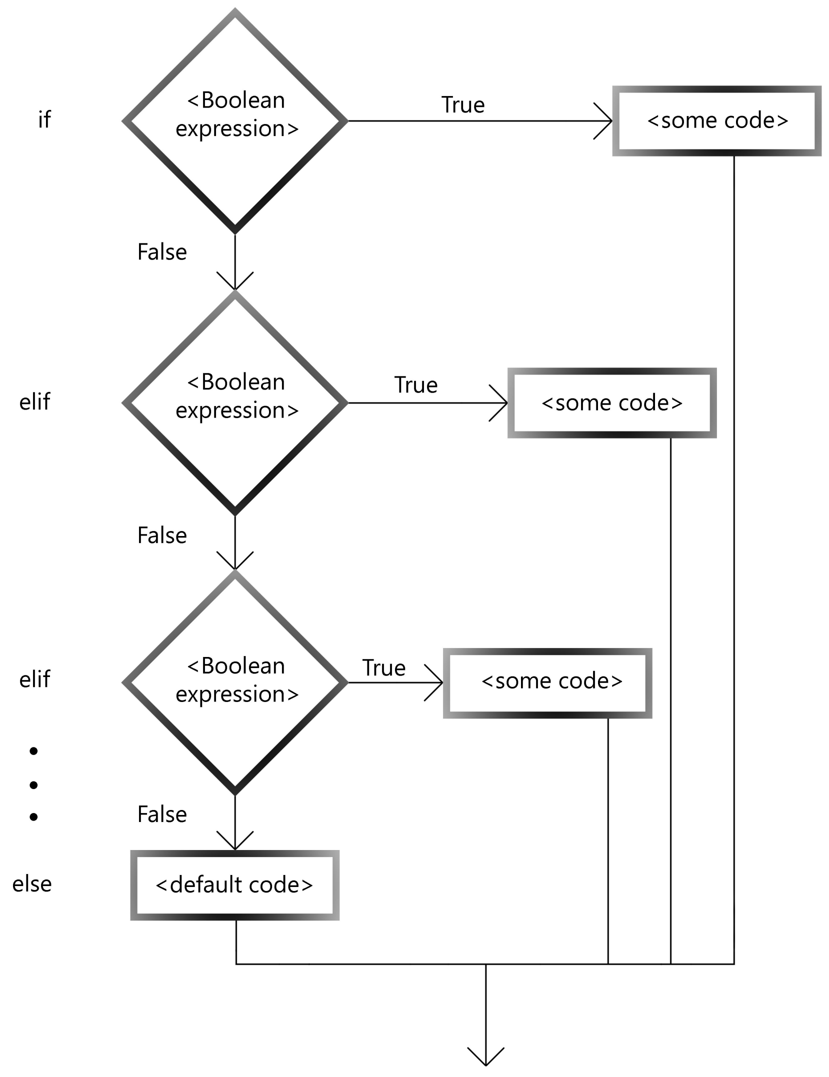
```
print(createHeader('Joe Smith', 'm'))
print(createHeader('Susan Jones', 'f'))
print(createHeader('Henry Jones', 'm'))
print(createHeader('Chris Smith', '?'))
```

In this example, you really wanted to check for 3 different cases. It turns out that often, you need to check for three or four or five, or any number of cases. When trying to write code for this, you could indent after every False case, just as we did here where the gender was not male. But when you start having many cases to check for, the indenting becomes very difficult to read.

Python has solved this problem by giving us a way to check for multiple different cases without having to continuously indent. Let me introduce an addition to the 'if' statement called the 'elif' statement. 'elif' is short for 'else if'

elif statement:

+++ To understand how this works, let me first show you a flowchart:



The basic idea is that you can check for multiple conditions, and when there is a match, that is, a boolean expression

evaluates to True, then the code associated with that condition is executed.
++++
Python syntax of if/elif/else:

```
if <boolean expression>:
    <some code>
elif <boolean expression>:
    <some code>
elif <boolean expression>:
    <some code>
# as many elif’s as you need ...
else:
    <some default code>
```

++++ Now, let me show what our createHeader program would look like using if/elif/else:

```
def createHeader(fullName , gender):
        if gender == ‘m’:
                title = ‘Mr. ’
        elif gender == ‘f’:
                title = ‘Ms. ’
        else:    #not sure, could be male or female
                title = ‘Mr. or Ms. ’
        header = ‘Dear ’ + title + ' ' +fullName
        return header


print(createHeader(‘Joe Smith’, ‘m’))
print(createHeader(‘Susan Jones’, ‘f’))
print(createHeader(‘Chris Smith’, ‘?’))
```

++++ Build program to figure out what to wear

```
def whatToWear(temperature):
        if temperature > 90:
                clothes = ‘swim suit’
        elif temperature > 70:
                clothes = ‘shorts’
        elif temperature > 50:
                clothes = ‘long pants’
        else:
                clothes = ‘thermal underwear and long pants’

        return clothes

print(‘Put on’, whatToWear(100))
print(‘Put on’,  whatToWear(40))
print(‘Put on’,  whatToWear(71))
```

While we will not going into this level in this class, but here is a great example of where you might use this. Imagine you are writing a game program. The program could respond to a number of different keys on the keyboard. For example, the left, right, up, and down arrows could move a character on the screen in that direction, or maybe pressing the space bar could mean shoot. To code something like this, you would use a series of if/elif/else comparisons to see what key was pressed, and you would call a different function to do some action based the key that was pressed.

If there are any people who know other languages like C or Java, you might recognize this as a ‘switch/case’ statement.

A (simplistic version of) a real-world example - grading:

++++

Grading program example:

```
# Convert score to letter grade:
def letterGrade(score):
        if score >= 90:
                letter = ‘A’
        elif score >= 80:
                letter = ‘B’
        elif score >= 70:
                letter = ‘C’
        elif score >= 60
                letter = ‘D’
        else:
```

```
        letter = ‘F’  #fall through or default case
    return letter


grade1 = letterGrade(75)
print(grade1)
grade2 = letterGrage(82)
print(grade2)
print(letterGrade(95))  # call and print in one statement
```

Next, we will work through a number of small sample programs, starting with a very easy one, and working our way through more complex ones. As we try to solve these challenges, we will have to start thinking harder about our approach to solving the more difficult ones.

+++++

DEF: <u>Algorithm</u> – a series of steps to solve a problem

Try to think through your problem in English before writing any code.

Think through the approach in English before writing any code:

After writing the function, write sample code to test the function and print results.

Absolute Value

Create a function called ‘absoluteValue’. It will be passed one numeric (integer or float) parameter. An absolute value is the (positive) distance from zero on a number line. That is, absoluteValue should always return the positive value of the original number.

While Python has a built in function for this, please do not use it in this assignment.

(The function should NOT print anything.)

Write main program code to call the function with test values, then print the returned results of each call:

yyy

____

For extra practice:

Allow the user to enter a number, use it in your call to the function, and print the result.

Be sure to use different variable names for the user’s input, and the parameters used in your function.

Square

Create a function called ‘isSquare’. It will be passed two parameters, length and width. isSquare should return:

True if the sides represent a square

False if the sides do not represent a square.

(The function should NOT print anything.)

Write main program code to call the function with test values for the sides, and print the following based on the returned result:

True

OR

False

____

For extra practice:

Allow the user to enter values, use them in your function call, and print based on the results.

Be sure to use different variable names for the user’s input, and the parameters used in your function.

Negative, Positive, Zero

Create a function called ‘negativePositiveZero’. It will be passed one numeric (integer or float) parameter. negativePositiveZero’should return one string value:
‘negative’ if the number negative
‘positive’ if the number is positive.
‘zero’ if the number is zero

(The function should NOT print anything.)

Write main program code to call the function with test values, and print the returned results.

```
positive

OR

negative

OR

zero
```

For extra practice:

Allow the user to enter a value, use that value in your function call, then print the results.

Be sure to use different variable names for the user’s input, and the parameters used in your function.

Even

Create a function called ‘isEven’. It will be passed one numeric (integer) parameter. isEven should return a Boolean:
    True if the number is even (...-6, -4, -2, 0, 2, 4, 6, ...)
    False if the number is odd (... -5, -3, -1, 1, 3, 5, 7, ...)

(The function should NOT print anything.)

Write main program code to call the function with test values, and print the following based the results of each call:

True

OR

False

____

For extra practice:

Allow the user to enter a value, and use that value in a call to your function, print results as above.

Be sure to use different variable names for the user’s input, and the parameters used in your function.

Rectangle

Create a function called 'isRectangle'. It will be passed four parameters representing the length of all four sides – in the order of left, top, right, and bottom. isRectangle should return:

True if the sides represent a rectangle
False if the sides do not represent a rectangle.

(The function should NOT print anything.)

Write main program code to call the function with test values for the sides, and print the following based on the returned result:

True

OR

False

____

For extra practice:

Allow the user to enter values, use them in your function call, and print based on the results.

Be sure to use different variable names for the user's input, and the parameters used in your function.

For a real challenge, see if you can code this where the values of the sides as sent in ANY order!

Conditional logic:

So far, all conditions have been using a single operator (e.g., ==, != , >, <, etc.) However, boolean expressions can be as simple or as complex as you want them to be. Boolean expressions can contain multiple checks using the keywords “and”, “or” and “not”.

Logical <u>not</u> operator:

We’ll start with the easiest one, the not operator. The not operator takes in a boolean value, and if the value is False, it changes the value to True. If the value is True, it changes the value to False. It is often used in if statements to make things clear.

Examples:

```
if not open:
    # closed

if not broken:
    # working

if not(width == length):
    # not a square


# can use it in an assignment statement:
alive = not dead
```

```
# maybe reverse the output of a function:
isOdd = not isEven(value)
```

Here is a “Truth table” for the not operator, very simple:

```
NOT:
Input | Output
__________

T     |   F
F     |   T
```

Logical and operator:

Also, can check multiple conditions by adding “and”

Examples

```
if (age > 12) and (height > 48):
    # OK to get on roller coaster

if (nDollars > 4) and atInAndOutBurger:
        # can buy double double and fries


if inThisClass and studyingHard and doingHomework:
    # you will do well in this class
```

```
if (x > 5) and (x < 10):
    # x is greater than 5 and less than 10
```

AND Truth table:

A and B		Result
F	F	F
F	T	F
T	F	F
T	T	T

Notice that you get a False in every case except when both inputs are True. Another way to think about this is, the boolean will be True is ALL input values are True

Logical <u>or</u> operator:

Examples:

```
if (nDollars > 4) or dateIsPaying:
    # can buy ice cream sundae

if (studyingHoursPerDay > 4) or
```

```
        payOffTeacherForGoodGrade:
    # you will do well in class (JOKE!)

if (age > 65) or disabled:
    # can get government benefits
```

OR Truth table:

A or B		Result
F	F	F
F	T	T
T	F	T
T	T	T

If both inputs are False, the result if False. If either or both inputs are true, then the result is true. Generates a True if ANY of the inputs are True.

Now the check for a rectangle can be written much simpler:

```
def isRect(side1, side2, side3, side4):
    if (side1 == side2) and (side3 == side4):
        return True
    else:
        return False
```

Combination of many operators:

```
if not inJail and location == ‘Silicon Valley’ and cash >=
1000000 or haveHighPayingJob and downPayment >= 90000:
    # Can buy a house in Silicon Valley
```

But what operations happen in what order?

Let’s force the order of operations by adding parentheses.

```
if (not inJail) and
   (location == ‘Silicon Valley’) and
   ((cash >= 1000000) or
        (highPayingJob and (downPayment >= 90000))):

# Can buy a house in Silicon Valley
```

[Optional extra practice]

Function to see if a value is between two other values:

```
def isBetween(value, low, high):
  if (value >= low) and (value <= high):
      return True
  else:
      return False
```

Function to see if length, width, height is a cube:

```
def isCube(length, width, height):
  if (length == width) and (width == height):
      return True
  else:
      return False
```

Program to determine shipping costs. Given country and total purchased, build a program based on the following:

[external sample file]

US:

<= 50	6.25
<= 100	9.50
<= 150	12.75
otherwise	15.00

Canada:

<= 50	8.25
<= 100	12.50
<= 150	18.00
otherwise	25.00

Do not ship to any other country

```
# Shipping expense

NOT_YET = ‘Not yet’

def calculateShipping(country, nWidgets):

    if (country == 'USA') or (country == 'US') or (country ==
'United States'):
        if nWidgets < 50:
            shippingCost = 6.25
        elif nWidgets < 100:
            shippingCost = 9.50
        elif nWidgets < 150:
            shippingCost =12.75
        else:
            shippingCost = 15.00

    elif country == 'Canada':
        if nWidgets < 50:
            shippingCost = 8.25
        elif nWidgets < 100:
            shippingCost = 12.50
        elif nWidgets < 150:
            shippingCost = 18.00
        else:
            shippingCost = 25.00

    else:
        shippingCost = NOT_YET  #  special value to say that we
don't ship to this country

    return shippingCost
```

```
# Get user input then call the above function

userCountry = input('What country are you shipping to? ')
countOfWidgets = input('How many widgets? ')

#convert to int
countOfWidgets = int(countOfWidgets)

#call the function to see how much it will cost to ship
amountForShipping = calculateShipping(userCountry,
countOfWidgets)
if amountForShipping == NOT_YET:
    print('Sorry, we do not ship to', userCountry)
else:
    print('It will cost $', amountForShipping, 'to ship your
package')
```