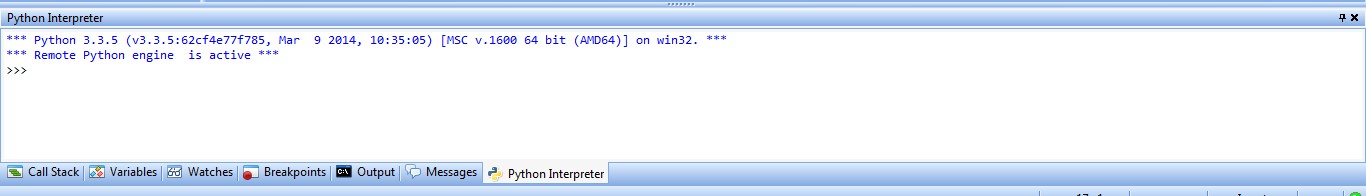
|  |  |
| --- | --- |
| **Worksheet 5: Flow of Control**  (Chapter 3 of Dawson) | Name: **Lex Baker**  Date: **9/20/22** |

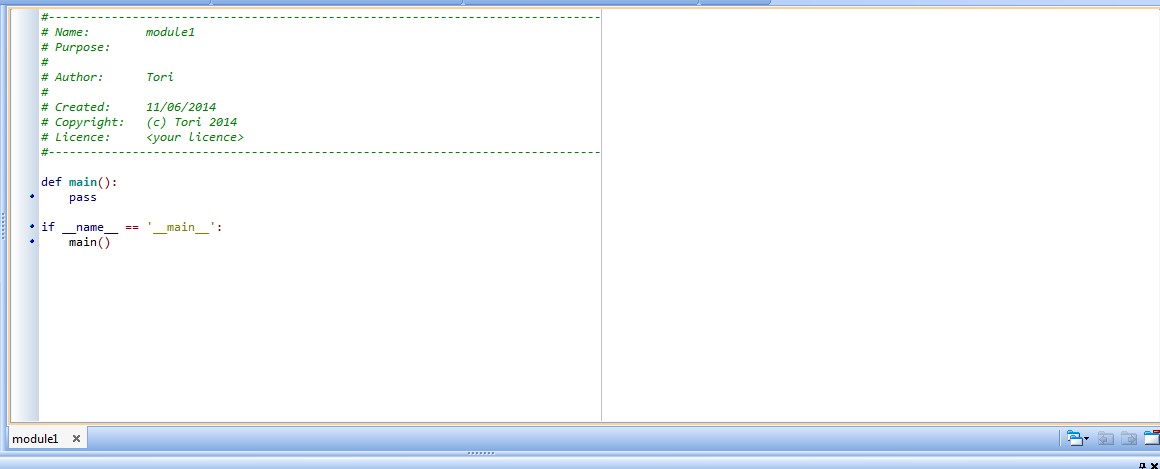
Before beginning to talk about the flow of control in programs, we will explore a different IDE known as PyScripter. This has a lot of helpful features that are much nicer than IDLE, but it's also a bit more complicated. The basic idea is the same.

Open up PyScripter. The ***Python Interpreter*** is the window at the bottom that looks like this:



It is identical to the Python Shell in IDLE. You can type commands at the command prompt to run them immediately. When you use print inside a program in the editor, the printed statements will appear here. One difference is that when you use "input," instead of asking for the users input in the interpreter, an input dialogue box will appear. You will see all of this shortly.

When you open PyScripter, the majority of the screen will be taken up by the ***editor***. It will look like this:



Just like in IDLE, this is where you type multiple Python instructions at once. As you see, PyScripter creates a header for you as well as an outline for basic code. You will usually want to delete all of the lines shown above and write you own code. Go ahead, delete all of the prewritten code and add back in a header with your name, date, and worksheet number. To run code in the editor, either click the green arrow: or hit ctrl-F9.

**Flow of control** refers to the “order” in which the statements of your program are executed. In this context, “order” refers not just to which statements are executed before others, but also which are ignored and which are repeated.

# 1. Boolean variables

Decisions about whether or not to execute a group of statements or how long to keep repeating a group of statements are based on a condition being true of false. To facilitate these decisions all programming languages have a variable type that can only take on one of two values – True or False. These variables are called ***Boolean variables*** or ***Booleans***.

**a.** The simplest way to create a Boolean variable in Python is with an assignment statement. Enter the following commands into the Python Interpreter and write down the output.

>>> friday = True;

>>> print("friday is type: ", type(friday));

>>> sunny = False;

>>> print("Is it sunny today?", sunny);

You MUSTcapitalize the first letters in True and False to have Python recognize them as keywords and create a Boolean variable. Try this:

>>> hawks\_fan = true;

True and False are reserved keywords (note that they turn orange). true and false are not reserved for anything. The failed line above went looking for a variable named true in the hope of assigning its value to the variable hawks\_fan, but it could not find any variables named true, so it threw an error.

# 2. Boolean operators

The operators +, \*, /, \*\* etc. combine int and float variable to create new int/floats. There are

three operators that work with Boolean variables: and, or, not These operators are also referred to as **logical operators**.

1. Create a new program file and type the statements below into the file. Run and re-run the code, changing the values of tired, caffeinated, and healthy each time. ##Exploring Boolean variables and logical operators tired = True; caffeinated = True; healthy = False;

ready\_to\_learn = caffeinated or (not tired); excited\_to\_learn = caffeinated and (not tired); go\_home = tired and (not healthy);

print("Are you ready to learn? ", ready\_to\_learn); print("Are you EXCITED to learn!!? ", excited\_to\_learn); print("Should you go home? ", go\_home);

1. Write a brief explanation of what each of these operators do.

**and: Takes two expressions and returns True only if both evaluate to True**

**or: Takes two expressions and returns True as long as one evaluates to True**

**not: Takes two expressions and returns True as long as both evaluate to False**

# 3. Comparison operators

Expressions involving comparison operators compare two values and then evaluate to the Boolean value appropriate to the comparison.

1. Write expressions in the Shell to figure out what each of the 6 comparison operators (<, >, <=, >=, ==, !=) do.

**< returns True if the first value is less than the second value**

**> returns True if the first value is greater than the second value**

**<= returns True if the first value is greater than or equal to the second value**

**>= returns True if the first values is less than or equal to the second value**

**== returns True if the two values are equivalent**

**!= returns False if the two values are equivalent**

1. Enter these expressions in the interpreter and formulate some hypotheses about what is going on...

>>> 'A'<'B'

>>> 'a'<'A'

>>> 'Z'<'a'

>>> '#'<'!'

>>> 'cat'<'dog'

>>> 'cap'<'cat'

**Python is converting the letters to Unicode numbers, where the higher in the alphabet the letter is, the less it’s worth, and where all capital letters are less any lowercase letters.**

# 4. Simple conditional statement

Conditional statements (also called conditionals or if-statements) look at a condition and use it to decide whether or not to execute a group of statements. In the simplest implementation, a condition is used to “turn on” or “turn off” a group of statements. Go back to your file, comment out the previous code, and type in the lines below exactly as they appear. Run it with different inputs until you figure out exactly how the program works.

##Uses an if-statement to control execution of two print commands number\_donuts = int(input("How many donuts do you want? ")); less\_than\_half\_dozen = number\_donuts < 6;

print("Mmmmm...eating", number\_donuts, "donut(s) sounds yummy.") if less\_than\_half\_dozen:

print("You know you are not getting the best value.") print("If you get a half-dozen or more, you get a discount.") print("You know what sounds even better?", number\_donuts\*2, "donuts.");

The colon and the indentation are a required as part of the ***syntax of the if-statement*** in Python. Execution of indented lines that follow the : is controlled by the value of the condition less\_than\_a\_half\_dozen. The first non-indented line, and all lines thereafter, are not controlled by this condition. The : and the indenting are the **syntax** used by Python to **delimit the body of the if-statement.** Indent the final print() command and re-run the code to see the effect of indenting.

**Indenting a line tells Python that that line is a part of the above line, such as with function declarations or if statements. Other languages use brackets for this.**

# 5. if-else statement

The next-simplest implementation involves executing one group of statements if a condition is True and an alternative group if it is False. The keyword else tells Python which statements to execute if the initial condition is not met. Comment out your previous code and enter the code below. Play with it to see how the if-else statement works.

##Uses if-else to run one group of statements or another number = int(input("Enter an integer: ")); even\_number = number%2==0;

if even\_number:

print(number, "is even.");

print("Is",number,"also divisible by 4?",number%4==0); else:

print(number, "is odd."); print("Is",number,"also divisible by 3?",number%3==0); print("What a great number.");

# 6. if-elseif statement

Using the keyword elif in your if-statement makes it possible to check a variety of conditions in deciding which group of statements to run. Comment out the previous problems and type the program below into your file. Figure out what it does and how it works.

##Uses if-elif to sort through coffee possibilities cream = True; sugar = False;

if ((not cream) and (not sugar)):

preference = "black"; elif ((not cream) and sugar): preference = "sweet"; elif (cream and (not sugar)): preference = "white"; else:

preference = "regular"; print("You like your coffee " + preference);

**This program takes two variables, cream and sugar, and depending on whether they are True or False, returns your preference for coffee.**

# 7. The while-loop

A ***loop*** is a programming structure wherein a group or statements is repeated over and over again. In a while loop, the statements are executed and re-executed until the loop’s condition becomes False. Enter and run the following simple example.

##Uses a while loop to make Python "count" max\_number = 15; counter = 0; while (counter < max\_number):

print(counter); counter = counter+1;

print("Good thing I incremented the counter or I would have counted FOREVER!!");

Now try taking out the line counter = counter+1 and see what happens. Hitting ctrl-C will stop your program when you enter an infinite loop.

**You enter an infinite loop because the while loop always evaluates to True**

# 8. losing\_battle-bad.py

Here's some practice with modifying already existing code. Find the file losing\_battlebad.py on Haiku and save it on your computer with a different name. This example is taken from Dawson's book (the code for all of his files can be found online - the link is on Haiku).

1. Run the program and practice bailing out.

**~Cue spamming ctrl+c~**

1. Now read the program and explain why, in your own words, the program gets stuck in an infinite loop. (The detailed discussion on p. 68/69 of the book that steps through this program is really important. You MUST understand it and be able to make tables of your own where you hand-execute code. Look over this section if you have not already done so while trying to answer this question.)

**Because the while loop only stops if health equals exactly 0, so even though health continues towards negative infinity, it never equals zero, and the program never ends.**

1. Modify the expression that creates the condition so that the infinite loop does not happen.

**while health >= 0**

1. This program is far more exciting if there is a little unpredictability. Remember math.randint from the last worksheet? Modify the program such that the damage inflicted on the hero each time through the loop is a random integer between 0 and 4. Also modify it such that the number of trolls the hero fells with each swing is a random integer between 1 and 3. The program should otherwise work identically. Once you are done, you can compete against your friends to see who can defeat the most trolls.

**Modified the losing\_battle-bad.py file.**

1. **a.** Write a simple if-statement that prompts the user to enter y or n to say whether or not they are tired. If they are tired, have the program output a message about getting more sleep. Have the program end by printing some encouraging message for the day. The encouraging message should print regardless of whether or not they enter y of n.

* 1. Add an else-clause to the conditional that will print a compliment message about their good sleep habits if they are not tired. The encouraging message should still always be printed.

* 1. Add an elif-clause so that the program will mildly chastise the user if they enter in any text other than y or n. The encouraging message at the end should still always be printed.

* 1. Do a little string manipulation (see parts of chapter 2 in the book or documentation online) and logical operation so that the program will work effectively for user inputs of y, Y, yes and Yes or n, N, no and No.

**tired = input("Are you tired? (y/n)\n").lower()**

**if(tired == "y" or tired == "yes"):**

**print("You should try to go to bed earlier.")**

**elif (tired != "n" and tired != "no"):**

**print("You didn't enter y or n/yes or no...")**

**else:**

**print("Congratulations on getting enough sleep!")**

**print("I hope you have a wonderful day!")**

1. Prompt the user for three numbers and then use conditional statements to print them back out in ascending order. This is harder than you think!! Do not use the built in max() and min() functions, you cheater. Alternatively have the user enter three words and have your program print them out in alphabetical order.

**num1 = input("Enter the first number: ")**

**num2 = input("Enter the second number: ")**

**num3 = input("Enter the third number: ")**

**if(num1 <= num2 and num1 <= num3):**

**print(num1)**

**if(num2 <= num3):**

**print(num2)**

**print(num3)**

**else:**

**print(num3)**

**print(num2)**

**elif(num2 <= num1 and num2 <= num3):**

**print(num2)**

**if(num1 <= num3):**

**print(num1)**

**print(num3)**

**else:**

**print(num3)**

**print(num1)**

**else:**

**print(num3)**

**if(num1 <= num2):**

**print(num1)**

**print(num2)**

**else:**

**print(num2)**

**print(num1)**

## 11. Math with loops

1. Prompt the user for a positive integer. Use a while-loop to compute the sum of the integers from 1 to what they entered. At the end report the result.

**num = int(input("Enter a positive integer:\n"))**

**sum = 0**

**while(num > 0):**

**sum += num**

**num -= 1**

**print("sum =", sum)**

1. Add to this while-loop to also compute the factorial of the user’s number. Report both the sum and the factorial at the end.

**num = int(input("Enter a positive integer:\n"))**

**sum = 0**

**fact = 1**

**while(num > 0):**

**sum += num**

**fact \*= num**

**num -= 1**

**print("sum =", sum)**

**print("factorial =", fact)**

**Quick Check**

**12.** Have the user enter in 3 lengths for the sides of a triangle. Check to see if the triangle is possible. If it is, tell them the angles. If it isn’t, let them know.

**import math**

**s1 = float(input("Enter the first side length:\n"))**

**s2 = float(input("Enter the second side length:\n"))**

**s3 = float(input("Enter the third side length:\n"))**

**if(s1 + s2 <= s3 or s1 + s3 <= s2 or s2 + s3 <= s1):**

**print("The triangle you input is invalid")**

**else:**

**a1 = math.acos((s2\*\*2 + s3\*\*2 - s1\*\*2) / (2\*s2\*s3)) \* (180/math.pi)**

**a2 = math.acos((s1\*\*2 + s3\*\*2 - s2\*\*2) / (2\*s1\*s3)) \* (180/math.pi)**

**a3 = math.acos((s1\*\*2 + s2\*\*2 - s3\*\*2) / (2\*s1\*s2)) \* (180/math.pi)**

**print("The angles, in order, are:", end=" ")**

**print(a1, a2, a3, sep=", ")**