

Activity 1.3.3 Branching and Output

Introduction

|  |  |
| --- | --- |
| Computers can sometimes appear intelligent because they can make a decision. A program that lets you text from a mobile device might decide to autocorrect your text. A program to provide driving directions might decide you've gone the wrong way. All programming languages have a way to branch, executing one set of instructions or another, depending on a condition.  How is this done in *Python*® programming language? And how can we get output from a program to know what the program has decided? | Conditionals |

Procedure

1. Form pairs as directed by your teacher. Meet or greet each other to practice professional skills. Launch Canopy and open an editor window. Set the working directory for the iPython session and turn on session logging. Open a new file in the code editor and save it as JDoeJSmith\_1\_3\_3.py.

In []: %logstart -ort studentName\_1\_3\_3.log

In []: *# Jane Doe John Smith 1.3.3 iPython log*

1. In the previous activity, you learned that you can assign values to variables of different types. You learned that you can evaluate expressions and can define functions that return a value.

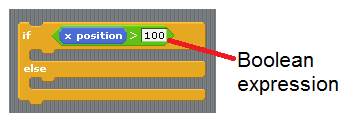
In []: a = 3

In []: a\*\*2

Out[]: 9

**Part I: Conditionals**

1. As you saw in Scratch™ programming language, computer programs can use if structures to make decisions. In an if structure, what gets executed depends on whether a **Boolean** expression, also known as a **conditional**, is true.



In []: a == 3 *# Boolean expression, also called a conditional*

Out[]: True

Note that the single = is used for assignment, while the double == is used for a Boolean expression. Other Boolean operators include >= (for ≥) and != (for ≠).

You can make **compound conditionals** by connecting Boolean expressions

with and, or, and not. In the following examples, remember that a is still 3.

In []: a+1 >= 2 and a\*\*2 != 5 *# compound conditional*

Out[]: True

The only two Boolean values are True and False.

Predict, try, discuss, and explain the output:

*# 3a. Prediction:* Out[]:  **True**

In []: a\*\*2 >= 9 and not a>3

Out[]:  **True**

*# If prediction was wrong, discuss and explain*

**\_**

*# 3b. Prediction:* Out[]:  **True**

In []: a+2 == 5 or a-1 != 3

Out[]:  **True**

In []: *# If prediction was wrong, discuss and explain***\_**

1. One situation where a programmer would use a compound conditional is when deciding whether a mouse click is within a certain region on the screen. For example, you might want your program to respond if a user clicks a rectangular shape, like a button or tab, as shown in the figure on the left.

|  |  |
| --- | --- |
|  |  |

Creating a program that uses mouse input is part of Lesson 1.5. For now, we’ll just assign x and y arbitrary values—let’s say (65,40), using the **assignment operator** =. These coordinates are shown by the red dot in the figure above on the right.

Unlike most languages, *Python* allows **multiple assignment** in a single line of code, as shown here:

In []: x, y = (65, 40)

In []: x

Out[]: 65

You might have noticed that the positive y-axis points down in the figure above. Computer graphics usually number the pixel rows from 0 at the top left and increase down and to the right across the screen.

The following expression is True when the red point ***(x, y)*** is in the blue area shown above.

In []: 50<x and x<80 and 30<=y and y<=45

Out[]: True

Write a compound conditional to test whether ***(x, y)*** is in the rectangle shown in the figures below. The conditional should be False since ***(x, y)*** is still bound to the values (65,40) from before.

|  |  |
| --- | --- |
|  |  |

In []: *# 4. Compound conditional*

In []: *x, y == (100, 110)*

Out[]: False

1. Assign (90,115) to ***x*** and ***y*** to match the figure above with the mouse cursor on the left and the red dot on the right. You can use the up arrow on your keyboard to go back in the iPython session history and reuse the compound conditional you typed in Step 4. You can press enter to execute the command again or modify it before pressing enter.

In []: x, y = (90, 115)

In []: *(arrow up to retrieve your compound conditional)*

Out[]: True

Part II: if–else Structures and the print() Function

1. An if structure causes a program to execute a block of code only if a Boolean expression is True.

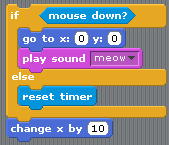
Optionally, an else block can be included. The else block of code will be executed only if the Boolean expression was False.

No matter whether the Boolean expression was True or False, the execution continues after the else block of the if–else structure. The function below illustrates the if–else structure.

The function also shows the print() function, which will print its arguments on the screen. The text inside the single quotation marks is called a string of characters and will be printed as-is, without the quotes.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **from \_\_future\_\_ import** print\_function*# use Python 3.0 printing*  **def** age\_limit\_output(age):  *'''Step 6a if-else example'''*  AGE\_LIMIT = 13 *# convention: use CAPS for constants*  **if** age < AGE\_LIMIT:  print(age, 'is below the age limit.')  **else**:  print(age, 'is old enough.')  print(' Minimum age is ', AGE\_LIMIT) |

The colons at the end of lines 3, 6, and 8 are required by def, if, and else key words. The indentation tells the *Python* interpreter which block of code belongs to that def, if, or else block. Just as Scratch grouped the code in if/else blocks, *Python* uses indentation to group code. Always use four spaces for each level of indentation. All such stylistic conventions can be found at [**http://www.python.org/dev/peps/pep-0008/#indentation**](http://www.python.org/dev/peps/pep-0008/#indentation)



When you try the code, notice that the output doesn’t come after an Out[]:. This is because print() sends output to the system out, which by default is set to be the screen. The print() command returns a null value, however. The return value of a function or expression is what iPython displays (also on the screen) after the Out[].

1. Copy the code above to the code editor. Execute the code, and then try the following input. Discuss the output.

In []: age\_limit\_output(10)

In []: age\_limit\_output(16)

1. Define a new function report\_grade(percent) that reports mastery if the argument percent is 80 or more. You can write the function as an additional function in the same file of code as before. The beginning of the new code is shown here, and the required output is shown below. Pair program, strategizing first.

|  |  |
| --- | --- |
| 12  13 | **def** report\_grade(percent):  **If percent >= 80:**  **"a grade of %s**  *'''Step 6b if-else'''* |
|  |  |

In []: report\_grade(79)

A grade of 79 does not indicate mastery.

Seek extra practice or help.

In []: report\_grade(85)

A grade of 85 percent indicates mastery.

Keep up the good work!

Part III. The in operator and an introduction to collections

The in operator is another Boolean operator, like ==. Since it is Boolean, it returns True or False. You can use it to see if an element is in an iterable. Iterables are built out of zero or more elements and include strings as well as other variable types like tuples and lists that are the subject of the next activity. A string is an iterable that is made of elements that are characters.

|  |  |  |
| --- | --- | --- |
| Symbol Name | Looks Like | Iterable |
| Quotation marks | 'letters' or "spaces" | string |
| Parentheses | ( e1, e2, e3 ) | tuple |
| Square brackets | [e1, e2, e3 ] | list |
| Curly braces | {key1:value1, key2:value2} | dictionary |

1. Examine the following examples using the in operator:

In []:'t' **in** 'string' *# includes lowercase t character*

Out[]: True

In []:'T' **in** 'string' *# case matters*

Out[]: False

In []: 3 **in** [1,2,3] *# this list contains the int 3*

Out[]: True

In []:'3' **in** [1,2,3] *# '3' is a string, different than 3*

Out[]: False

1. The following function vowel(letter) returns True if the letter is 'a', 'e', 'i', 'o', 'u', or any of their uppercase counterparts and returns False otherwise.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **def** vowel(letter):  vowels = 'aeiouAEIOU'  **if** letter in vowels:  return True  **else**:  return False  *# should check len(letter)==1* |

In your *Python* file for this activity, define a function letter\_in\_word(guess, word) that returns True if guess is a letter in word and returns False otherwise.

In []: letter\_in\_word('t', 'secret hangman phrase')

Out[]: True

1. In MasterMind, one player has a secret sequence of colored pegs. Another player tries to guess the sequence.

Define a function hint(color, secret) that takes two parameters: a string (representing a color) and a list of strings (representing a sequence of colors). The function should print a hint telling whether the color is in string.

In []: secret = ['red','red','yellow','yellow','black']

In []: hint('red', secret)

The color red IS in the secret sequence of colors.

In []: hint('green', secret)

The color green IS NOT in the secret sequence of colors.

Conclusion

1. Describe the relationship between blocks of code indented after the colon in if , elif, and else blocks.

**\_Ordered. If the last returns false the next runs**

1. There are many Boolean operators. List the ones you have learned about and name one more that you learn about by searching for Boolean operators on the Internet.

**\_and, or, not, without**

1. Steve and Latisha wrote this code:

|  |  |
| --- | --- |
| 21  22  23  24  25 | **if** check == 2:  print('Code complete.')  **else**:  print('Code complete.')  print('Not all systems are ready') |

Ira, Jayla, and Kendra are all saying it would be better to move lines 22 and 24 to a single line executing print ('Code complete.') just before line 21. These three students have different reasons for their opinions. Their reasons are below. Do you think each of them is right, wrong, or somewhere in between? Explain.

**\_Code only completes in check equals 2, so line 24 shouldn’t be there. Other than that the code is good**

Ira: “It would be better to have a single print statement because that code is going to happen no matter what. The program will run slower by having it there twice.”

**\_**

Jayla: “It would be better to have a single print statement because that code is going to happen no matter what. Later, if you want to change your program, you’re going to have to remember to change it in two places the way the code is now.”

**\_**

Kendra: “It would be better to have a single print statement because it is going to happen no matter what. That program would take up less memory if you just wrote it once.”

**\_**