

Activity 1.3.4 Nested Branching and Input

Introduction

|  |  |
| --- | --- |
| Most useful programs have a way to get input from the user, make a decision, and do different things depending on the input. Programs usually have a way to communicate output back to the user.  Think of a program or app you've used. What was the input? What was the output? Did the program's behavior depend on the input? | http://upload.wikimedia.org/wikipedia/commons/thumb/0/0a/QWERTY_keyboard.jpg/330px-QWERTY_keyboard.jpg |

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\_4.py.

In []: %logstart -ort studentName\_1\_3\_4.log

In []: *# Jane Doe John Smith 1.3.4 iPython log*

Part I. Nested if structures and testing

1. The if–else structures can be nested. The indentation tells the *Python*® interpreter what blocks of code should be skipped under what conditions. Paste the code below into your *Python* file. The line numbers will be different.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **def** food\_id(food):  ''' Returns categorization of food  food is a string  returns a string of categories  '''  *# The data*  fruits = ['apple', 'banana', 'orange']  citrus = ['orange']  starchy = ['banana', 'potato']  *# Check the category and report*  **if** food **in** fruits:  **if** food **in** citrus:  **return** 'Citrus, Fruit'  **else**:  **return** 'NOT Citrus, Fruit'  **else**:  **if** food **in** starchy:  **return** 'Starchy, NOT Fruit'  **else**:  **return** 'NOT Starchy, NOT Fruit' |

In []: food\_id('apple')

'NOT Citrus, Fruit'

1. Did this return value result from line 15, 17, 20, or 22 (refer to line numbers shown above)?

**\_17**

1. Every input will cause only one of the following lines of code to be executed.
2. What input will cause line 15 to be executed?

**\_orange**

1. What input will cause line 17 to be executed?

**\_apple**

1. What input will cause line 20 to be executed?

**\_potato**

1. What input will cause line 22 to be executed?

**\_water**

1. Bananas are starchy, and the program "knows" it. Explain why line 20 will never result in bananas being reported as starchy.

**\_Because banan are FRUIT**

1. The example in the previous step shows one reason bugs can be difficult to track down. Just the job of getting the program to “fall into” all the blocks of code can be difficult, and bugs can hide for years in a rarely executed line of code in a large program. To create code with fewer bugs, developers use glass box testing. That means they create a test suite that will run through every block of code. Some programmers write their test suite first, an approach called test-driven design or Extreme Programming (XP).

Continuing in your *Python* file, complete this food\_id\_test() that calls food\_id() several times: once for each of the one-line blocks at lines 15, 17, 20, and 22.

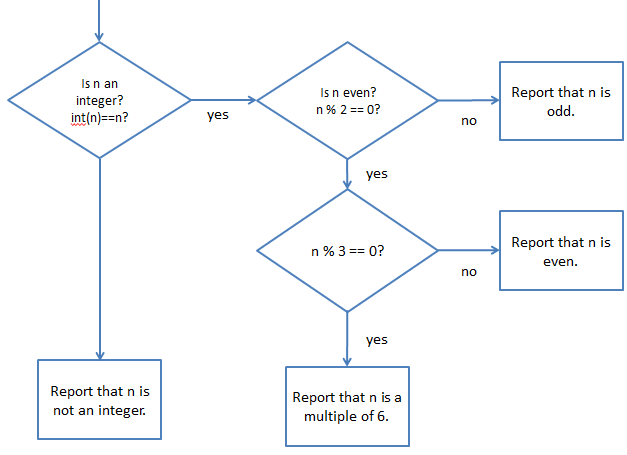
|  |  |
| --- | --- |
| 24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | def food\_id\_test():  ''' Unit test for food\_id  returns True if good, returns False and prints error if not good  '''  works = True  if food\_id('orange') != 'Citrus, Fruit':  works = 'orange bug in food id()'  if food\_id('banana') != 'NOT Citrus, Fruit':  works = 'banana bug in food\_id()'  *# Add tests so that all lines of code are visited during test*    if works == True:  print("All good!")  return True  else:  print(works)  return False |

In []: food\_id\_test()

All good!

Out[]: True

1. Define a function f(x) that implements this **flow chart**. A flow chart is another way to represent an algorithm; input and output are in rectangles, and branching decisions are in diamonds. The exercise illustrates the % operator, which identifies the remainder after division. As an example, 13 % 4 is 1, since 13÷ 4 is 3 remainder 1.



In []: f(12)

'The number is a multiple of 6.'

1. What set of test cases could you use to visit all the code?

**\_int(x) == x, x % 2 == 0, x % 3 == 0**

**Part II: The** raw\_input() **function, type casting, and print() from *Python* 3**

1. To get input from the user of a program, we normally use a graphical user interface (GUI). That is the subject of Lesson 1.3. Beginners often want a simple way to obtain text input. *Python* uses the raw\_input(prompt) command. It has some annoying behavior that we have to deal with for now — it always returns a string even when numeric type is appropriate. In addition iPython ignores Ctrl-C interrupts with raw\_input(), so infinite loops will require restarting the *Python* kernel. Finally the prompt doesn't appear until the user starts typing. That said, here’s how you use it:

In []: a = raw\_input('Give me a number between 5 and 6: ')

Give me a number between 5 and 6: 5.5

Even though the user typed a number, raw\_input() returned a string. You can see that as follows.

In []: a

Out[]: u'5.5'

In []: type(a)

Out[]: unicode

The variable a has a variable type that is a string. Keyboard input might be encoded as a unicode type, as shown above, or as a str type, but either way, it is a string of characters. (Unicode is a set of characters that includes all of the world’s written languages. It is encoded with UTF-8, an extension of ASCII. The u in u'5' indicates that the string returned by the raw\_input() command is a Unicode string.)

To use numeric values from the input, you have to turn the string into an int or a float. This will raise an error if the user didn’t provide an int or a float. There are commands – not covered in this course – that catch the error so that it doesn’t continue up to the *Python* interpreter and halt the program. For now, however, we can live with an error if the user does something unexpected.

To convert from a string to a number, you can use the int() function or the float() function. Forcing a value to be converted to a particular type is called type casting. Continuing from a being '5.5' above,

In []: int(a)

ValueError: invalid literal for int() with base 10: '5.5'

In []: float(a)

Out[]: 5.5

In []: int(float(a))

Out[]: 5

You can also type cast a number into a string:

In []: b = 6

In []:a + b

TypeError: cannot concatenate 'str' and 'int' objects

In []:a + str(b)

Out[]: '5.56'

In []:float(a) + b

Out[]: 11.5

Explain the difference between + as concatenation and + as numeric addition.

**\_**

1. The following code picks a random number between 1 and 4 (inclusive, meaning it includes both 1 and 4) and lets the user guess once. In part b below, you will modify the program so that it indicates whether the user guessed too low, too high, or correctly.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **from \_\_future\_\_ import** print\_function*# must be first in file*  **import** **random**  **def** guess\_once():  secret = random.randint(1, 4)  print('I have a number between 1 and 4.')  guess = int(raw\_input('Guess: '))  **if** guess != secret:  print('Wrong, my number is ', secret, '.', sep='')  **else**:  print('Right, my number is', guess, end='!\n') |

In []: guess\_once()

I have a number between 1 and 4 inclusive.

Guess: 3

Right, my number is 3!

In []: guess\_once()

I have a number between 1 and 4 inclusive.

Guess: 3

Wrong, my number is 4.

In line 9, print('Wrong, my number is ', secret, '.', sep='') has four arguments: three strings and a keyword=value pair. This is print(s1, s2, s3, sep=''). If the sep='' were not there, this would print the three strings separated by spaces. The separator is a space—by default—for the output of print(), but the function offers a keyword for setting the separator to a different value. The argument sep='' sets it to the null string, i.e., the empty string.

You’ll learn about the random module in the next activity.

|  |  |
| --- | --- |
| 2  5 | **import** **random**  secret = random.randint(1, 4) *# randint() picks including endpts* |

1. Explain how line 11 works, using the explanation of line 9 as a model.

**\_It is concatenating the strings to make a grammatically correct and comprehensible statement.**

1. Modify the program to provide output as shown below.

In []: guess\_once()

I have a number between 1 and 4 inclusive.

Guess: 3

Too low - my number was 4!

In []: guess\_once()

I have a number between 1 and 4 inclusive.

Guess: 3

Too high, my number was 2!

In []: guess\_once()

I have a number between 1 and 4 inclusive.

Guess: 1

Right on! I was number 1!

1. Create a function quiz\_decimal(low, high) that asks the user for a number between low and high and tells them whether they succeeded.

In []: quiz\_decimal(4, 4.1)

Type a number between 4 and 4.1:

4.5

No, 4.5 is greater than 4.1

In []: quiz\_decimal(4, 4.1)

Type a number between 4 and 4.1:

4.05

Good! 4 < 4.05 < 4.1

Conclusion

1. What is the relationship between if-structures and glass box testing?

**\_Glass box testing is the method used to find bugs in a code, which uses if structures to narrow the possibilities**

1. Nested if-else structures can contain many blocks of code. How many of those blocks of code might be executed?

**\_However many you need to cover every possible outcome in a program**

1. What does a test suite do, and why do you think programmers often write test suites first, before they've even written the functions that will be tested?

**\_It helps programmers make sure that the code is working and lets them seeeeeeee**