INTPROG
Introduction to Programming

moodle.port.ac.uk

Practical Worksheet P07: Using Booleans and While Loops

Introduction

This worksheet covers the further use of control structures (decisions and loops); in particular it introduces `while` loops. It also covers more advanced Boolean expressions that include the use of the Boolean operators. The main purpose of the practical exercises is to give you further practice in developing code that uses different control structures.

Work through the worksheet at your own pace and, as always, make sure you study each segment of code and try to predict its effects before you enter it, and afterwards make sure you understand what it has done and why. Furthermore, feel free to experiment with the shell until you fully understand each concept.

Simple while loops

Try the following code using Pyzo/IEP's shell:

```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

Make sure you understand what is happening with this `while` loop. Notice that it produces the same output as the following `for` loop:

```
for i in range(10):
    print(i)
```

Which of the loops do you think is better for producing this output, and why? Now try the following code:

```
i = 1
while i < 1000:
    print(i)
    i = i * 2
```

Again, ensure you understand what is happening here before moving on. Could you write a `for` loop to perform the same task and, if so, do you think it would be more readable than this `while` loop?

The following code illustrates a common error made when using `while` loops. After entering it, you will get a continuous stream of 0s outputted to the screen, and you'll need to "kill" the execution by pressing the yellow (lightning) interrupt button at the top of the Pyzo/IEP window.

```
i = 0
while i < 10:
    print(i)
```

Make sure you understand why this has happened. We often call such execution an ***infinite***

loop, and infinite loops are almost always the result of a fault (bug) in a program.

Using while loops to process user input

Copy the addnumbers.py file from Moodle into your own file-space. This file contains the functions from the lecture that output the sum of a sequence of numbers entered at the keyboard. Execute the functions, study the code, and make sure you understand how they work.

String testing methods

We saw last week that we can use the operators ==, <, > etc. for comparing two strings; e.g.,

```
name = "Sam"
name == "John"
name > "John"
```

There are also some string methods that test whether strings meet certain conditions. Try, for example:

```
myString = "all lower case"
myUniv = "University Of Portsmouth"
myNumber = "178"
myString.islower()
myUniv.islower()
myUniv.isupper()
"HELLO".isupper()
myUniv.istitle()
myString.istitle()
myString.isalpha()
"abcXYZ".isalpha()
myNumber.isalpha()
myNumber.isdigit()
myString.isdigit()
"321 hello".isdigit()
```

Further Boolean expressions

In Practical Worksheet P06, we introduced the Boolean operators `and` & `or` for combining Boolean expressions. The way these operators behave can be fully explained by entering all possible values for their operands (arguments). i.e.:

```
True and True
True and False
False and True
False and False

True or True
True or False
False or True
False or False
```

We see that `and` gives `True` only when both operands are `True`, and `or` gives `True` when

one or more of its operands are `True`. There is also a `not` operator:

```
not True
not False
```

that negates (flips) its operand value. These operators can be combined to give more complex Boolean expressions. For example, suppose that men receive a pension at 66 and women at 63. Then the following function works out whether someone receives a pension, assuming the gender parameter has the value `"m"` or `"f"`:

```
def isPensionable(gender, age):
    if (gender == "m" and age >= 66) or (gender == "f" and age >= 63):
        print("Yes")
    else:
        print("No, not yet")
```

Notice that we use parentheses here to make the meaning of the condition clear, although (due to the operators' precedence rules) they are not actually required in this case. Try, for example:

```
isPensionable("m", 64)
isPensionable("f", 64)
```

Using while loops to validate user input

Try the following code which is a typical example of how a while loop can be used to obtain valid input from the user (in this case we wish to ensure that a positive number is entered):

```
def getPositiveNumber1():
    number = 0
    while number <= 0:
        number = eval(input("Enter a positive number: "))
    return number
```

(Note the use of the first (assignment) statement which makes the loop condition `True`, thus forcing the loop body to be executed at least once.) There is an alternative form for the same task that involves a `break` statement:

```
def getPositiveNumber2():
    while True:
        number = eval(input("Enter a positive number: "))
        if number > 0:
            break
    return number
```

Note the difference here. There is no need for an initialisation statement in the second function since the loop condition is always `True`. The `break` statement is used to terminate (i.e. "break out of") the loop. The first function is probably preferable here, since it is easier to read. However, look at the `getString1` and `getString2` functions in the readstrings.py file from Moodle. These are more realistic examples of input validation since they provide feedback to the user if they input an invalid value. You might consider the `getString2` function (that uses the "loop-and-a-half" pattern with a `break`) to be the simpler function in this case.

Study these code segments and functions until you fully understand them — code for validating user input is extremely common and very important.

Programming exercises

When attempting this week's exercises, try to make sure that your solutions make good use of:

- control structures & Boolean expressions: it is normal for there to be several solutions to any given problem – try to write the simplest, most readable function you can.
- functions: the solutions to some of the later exercises may benefit from the use of extra functions to perform sub-tasks. Furthermore, you may already have solved some of these tasks for previous worksheets. In this case, you may access the function by importing the module file (e.g. `from pract05 import *`).

Copy the file pract07.py from Moodle into your own file-space. Your solutions to this week's exercises should be added to your copy of this file, and this should be made available for you to show to one of us in next week's practical session for feedback.

1. Write a `getName` function that reads a person's name from the user. Assume that a valid name is any string of alphabetic characters only, such as "Jenny". If the user enters an invalid name, the function should continue to ask the user for a name until she/he enters a valid one. Once a valid name has been entered, this should be returned. (Hint: use the `isalpha` method discussed above.)

2. The file pract07.py includes an incomplete `trafficLights` function that draws a set of traffic lights, with all lights off (i.e. black). Fill in the body of the while loop at the bottom of this function so that it will continuously cycle through the standard red $\rightarrow$ red/amber $\rightarrow$ green $\rightarrow$ amber $\rightarrow$ red sequence, simulating a real set of traffic lights. Use "yellow" for amber, and add realistic delays between light changes using the sleep function from the time module. E.g.,

   ```
   import time
   time.sleep(5)
   ```

   will cause a delay of 5 seconds.

3. In worksheet P06, you wrote a `calculateGrade` function that returned a grade for a pupil's coursework based on a mark. Write a function `gradeCoursework` that asks the user for a mark and, using a call to `calculateGrade`, displays a "The pupil achieved a grade of ..." message including the grade achieved. The function should only display a grade for valid marks (i.e. between 0 and 20) – if the user enters an invalid mark, they should be re-prompted until they enter a valid one.

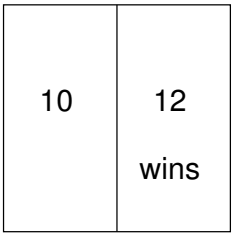
4. Write an `orderPrice` function that works out the price of an order of goods. The function should repeatedly ask the user for (i) the unit price of a product in the order, (ii) the quantity of that product in the order, and (iii) whether there are any more products in the order. When the user has completed entering prices & quantities, the function should output a message containing the total order price to 2 decimal places.

5. Write a `clickableEye` function which draws a brown eye of radius 100 within a graphics window of sufficient size. The function should then respond to each user click on the eye by displaying (underneath the eye) the name of the part of the eye clicked on (i.e. one of "pupil", "iris", "sclera" (or "white")). The user should be able to finish (i.e. close the window) by clicking on any point outside the eye. (Hint: use your `drawBrownEye` and `distanceBetweenPoints` functions from pract05.py.)

6. The pract07.py file contains functions `fahrenheit2Celsius` and `celsius2Fahrenheit` for converting between temperature units. Using calls to these two functions, write a `temperatureConverter` function that provides a text-based interface which allows the user to (repeatedly) convert temperature values until he/she wishes to stop. The user should be asked which way the conversion should be performed separately for each conversion.

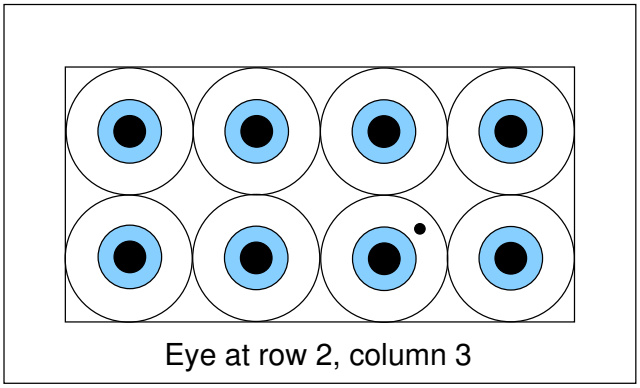
7. Write a `guessTheNumber` function. The function should generate a random number between 1 and 100 (using `randint` from the `random` module) and then allow the user to guess the number. After each incorrect guess, it should display "Too high" or "Too low" as appropriate. If the user guesses the number within seven guesses, it should display a "You win!" message saying how many guesses it took. After seven incorrect guesses, the function should display a "You lose! – the number was . . . " message.

8. Write a `tableTennisScorer` function that allows the user to keep track of the points of two players in a game of table tennis. In table tennis, the first player to reach 11 points wins the game; however, a game must be won by at least a two point margin. The points for the players should be displayed on two halves of a graphics window, and the user clicks anywhere on the appropriate side to increment a player's score. As soon as one player has won, a "wins" message should appear on that side; e.g.:



   The next click of the mouse should close the window.

9. [harder] Write a function `clickableBoxOfEyes` that takes two parameters `rows` and `columns`, and displays a `rows` × `columns` grid of blue eyes (all of radius 50) within a box (rectangle). There should be a border of size 50 between the box and the edge of the window. For each click of the mouse inside the box, the function should behave as follows: if the click is on an eye, the row and column of that eye should be displayed in the space below the box, for example as shown below given the click denoted by the dot (note that row and column numbers should begin at 1):



   If the user clicks within the box but not on an eye, the displayed message should be "Between eyes". The window should close when the user clicks outside the box.

10. [harder] Write a `findTheCircle` function to play a simple game. This should start by displaying a graphics window, and creating (but not displaying) a circle of radius 30 at a random position (use the `randint` function from the `random` module). The user should then have 10 attempts at locating the circle (by clicking on the graphics window). Each time (except the first) the user misses the circle, a "getting closer" or "getting further away" message should be displayed (depending on the position of the current and previous clicks). If the user manages to find the circle (by clicking within its circumference), then the circle should be displayed and the user given some points: 10 points for finding it with the first click, down to 1 point for finding it with the 10th click. The game then restarts. However, each time the game restarts the circle should be given a new random position and its radius reduced by 10%. The game ends when the user fails to find the circle within 10 clicks. The total number of points scored should then be displayed.

Written exercises

The aim of these exercises is to help to ensure that you understand Boolean expressions involving the Boolean operators (`and`, `or` and `not`), and can trace through the execution of for and while loops.

For and while loops

How many times will each of the following code fragments output the string "hello"?

1.
```
for i in range(10):
    print("hello")
```

2.
```
for i in range(10):
    for j in range(1, 5):
        print("hello")
```

3.
```
i = 1
while i < 10:
    print("hello")
    i = i + 1
```

4.
```
i = 1
while i <= 10:
    print("hello")
    i = i + 1
```

5.

```
i = 1
while i >= 10:
    print("hello")
    i = i + 1
```

6.

```
i = 1
while i < 10:
    print("hello")
```

Boolean expressions

We can make larger Boolean expressions from smaller ones using `or`, `and` & `not`. The operator precedence is as follows: `not` is higher than `and`; `and` is higher than `or`.

Evaluate each of the following Boolean expressions with respect to the two variables:

x | 3 | y | 2.65

7. `x > 1 and y < 10`
8. `x != 3 and y != 4`
9. `y > 2.4 or x < 2`
10. `x == 3 or x > 7 and y < 2`
11. `not x == 3 or y > 2`

Consider following code:

```
x = eval(input("Enter a number: "))
b = x == 10
for i in range(5):
    print(b)
    b = not b
```

12. What is the output when the user enters 10?
13. What is the output when the user enters 7?

Understanding more complex loops

Write down exactly what the output is for each of the following three functions that use `for` loops, given appropriate input values of your own choice.

For example, the function:

```
def mysteryA():
    number = eval(input("Enter a number: "))
    for i in range(number):
        print(i, end="")
```

on input value 5, gives the output:

```
01234
```

14.

```
def mystery1():
    string = input("Enter a string: ")
    number = eval(input("Enter a number: "))
    for ch in string:
        print(ch * number, end="")
```

15.

```
def mystery2():
    string = input("Enter a string: ")
    for i in range(len(string)):
        print(" " * i + string[i])
```

16.

```
def mystery3():
    number = input("Enter a number: ")
    x = 0
    for y in number:
        x = x + int(y)
    print(x)
```

17. What does the following function output? Try tracing the call `mystery4(3)`.

```
def mystery4(n):
   for i in range(n):
       for j in range(i+1):
           print("#", end="")
       print()
```