INTPROG
Introduction to Programming

moodle.port.ac.uk

Practical Worksheet P06: If Statements and For Loops

Introduction

This worksheet is intended to introduce you to simple Boolean expressions and decision structures, to review for loops, and to develop your algorithm development skills. 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.

Two of this week's exercises (9 and 10) provide a starting point for the Python assignment which will be distributed in week 9.

Simple Boolean expressions

Any decision a program makes is based on testing whether a certain ***condition*** is true or false. Conditions are examples of Boolean expressions (i.e. expressions of the Boolean data type). Let's investigate this type by experimenting with some simple Boolean expressions using Pyzo/IEP's shell:

```
True
False
type(True)
type(False)
x = 19
y = 7
x > 15
x < y
x <= 19
x <= 18
x + y <= 30
x == 19
45 != x
17 != x + 2
type(x > 19)
name = "Tim"
name == "Sam"
"Tom" != name
name < "Tom"
name < "Sam"
len(name) == 3
name[0] == "T"
```

Keep experimenting until you are sure you understand how Boolean expressions like these work, and you are aware of the meanings of the operators ==, !=, <, >, <=, and >=.

If statements

Enter the following function that includes an `if` statement:

```
def greet(name):
    print("Hello", name + ".")
    if len(name) > 20:
        print("That's a long name!")
```

Now try calling the function a few times with names of different lengths; e.g.:

```
greet("Sam Smith")
greet("Charles Philip Arthur George Jones")
```

Ensure you understand what is happening with each function call before moving on.

If-else statements

Enter the following function definition that includes an `if-else` statement:

```
def canYouVote(age):
    if age >= 18:
        print("You can vote")
    else:
        print("Sorry, you can't vote")
```

Now, invoke the function a few times with different numerical arguments:

```
canYouVote(23)
canYouVote(16)
canYouVote(18)
```

Again, ensure you understand what is happening here before moving on.

If-elif-else statements

Let's now define a function that includes an `if-elif-else` statement and that returns a value to the caller:

```
def getDegreeClass(mark):
    if mark >= 70:
        return "1st"
    elif mark >= 60:
        return "2i"
    elif mark >= 50:
        return "2ii"
    elif mark >= 40:
        return "3rd"
    else:
        return "Fail"
```

Now enter:

```
print("Your degree is", getDegreeClass(78))
print("Your degree is", getDegreeClass(45))
```

Repeat the call with different argument values until you understand the operation of the `if-elif-else` statement.

Using and & or

We can combine two Boolean expressions using the Boolean operators `and` and `or`, which we'll study in detail in Lecture P11. For now, let's see briefly some examples of their use:

```
x = 8
y = 3
x > 5 and y < 4
x == 4 and y < 4
x == 4 or y < 4
x == 4 or y > 5
```

Experiment with `and` & `or` for a while to try to fully understand their meanings.

Nested decisions and design issues

Copy the file dates.py from Moodle into your own file-space. This file contains three function definitions. The first one, `isLeapYear`, takes a year as a parameter and returns `True` or `False` depending on whether the year is a leap year or not. (Years divisible by 4 are leap years, except those that are divisible by 100. However, years that are divisible by 400 are also leap years. So, 2000 and 2004 were both leap years, but 2009 and 2100 are not.) Study the function definition to see how it works. Notice that it uses the `%` (remainder) operator to check if one value is divisible by another (e.g. `year % 4 == 0` is true if `year` is divisible by 4). Check that you understand the logic of the function. Test the function using the following calls:

```
isLeapYear(2000)
isLeapYear(2004)
isLeapYear(2009)
isLeapYear(2100)
```

Next in the dates.py file are two functions that give the number of days in a given month of a given year (where the month is expressed as a number between 1 and 12). Notice that both of these functions call the `isLeapYear` function when month is 2 (February) to find out if the given year is a leap year.

These two functions are equivalent to one another (they return the same value if given the same arguments). However, different design decisions have been made to write the two functions. In particular, `daysInMonth` considers the largest group of months (those that have 31 days) last so that they don't need to appear within a condition (they are dealt with in the else clause), whereas `overlyComplexDaysInMonth` considers February last, so the larger groups of months all have to be listed in the `if` and `elif` clauses.

This example is intended to illustrate that you should think carefully when designing decision structures. There are often several solutions to any given problem, and one solution is often simpler (better) than the others. The `daysInMonth` function could be re-written in a better way using lists, which we'll study later in the unit.

Simple For loops

Let's begin by reviewing how simple for loops work from earlier on in the unit. Most for loops use the built-in function `range`, so let's remind ourselves of what this does (using the shell):

```
range(10)
list(range(10))
```

```
len(range(10))
x = 5
range(x + 2)
list(range(x + 2))
len(range(x + 2))
```

We see that, given an integer argument `n`, `range` gives as an object of type `range`. A `range` object represents a range of values; in order to see all the values we need to pass this object to `list`. We see that the numbers within the range, and therefore in the resulting list, start with 0 and end with `n-1`. (Note that, assuming `n` is non-negative, the length of `range(n)` is `n`.)

Now, type the following:

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

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

(Note that you'll need to press return twice at the end of each loop to tell the shell that you've finished.) The first example shows us that the body of the loop (here, just a `print` statement) is executed the same number of times (10) as there are elements in `range(10)`. The second example shows that the variable `i` takes the value of each element of the list in turn, for each execution of the loop body. This can be seen even more clearly in the following for loops that don't use `range`. The first one loops through the elements of a list that hasn't been made using `range`:

```
for i in [1, 9, 7, 8]:
    print(i)
```

The next one loops through the elements of a string:

```
for ch in "Hello There":
    print(ch)
```

Note that each of these examples introduces a new loop variable–here `i` and `ch`. There is nothing special about these variables or their names (loop variable names should be chosen using the same criteria as any other variable). It is, however, important that you do not change the value of a loop variable within the body of a loop, as in:

```
for i in range(5): # Don't type in this example!
    i = i + 1
    print(i)
```

This leads to code that is difficult to understand (even though it might not actually cause an error).

Using extra arguments to range

If we want to count from 1 to 10 using a for loop and `range`, we might type:

```
for i in range(10):
    print(i + 1)
```

This code can be simplified slightly by using a second argument to range. Try:

```
list(range(1, 11))
```

```
list(range(-4, 20))
list(range(6, 2))
```

We see that `range(m, n)` gives a list that starts with `m` and ends at `n-1`. We can thus replace the above for loop by the slightly simpler:

```
for i in range(1, 11):
    print(i)
```

Finally, `range` can take a third argument. Try the following:

```
list(range(1, 10, 2))
list(range(4, 22, 3))
list(range(10, 0, -1))
```

We see that `range(m, n, s)` gives a range that starts with `m`, ends one value short of `n`, with a step of `s`. The final example shows how we can build a list with numbers in descending order.

Try entering, and then invoking the following function:

```
def countdown():
    for i in range(10, 0, -1):
        print(i, "...", end=" ")
    print("Blast Off!")
```

Experiment using `range` with different numbers of parameters until you are sure that you fully understand what it does and can predict its behaviour.

Nested for loops

Loops can be ***nested*** (appear one within another), as the following example demonstrates:

```
for i in range(3):
    print("Outer loop with i =", i)
    for j in range(4):
        print("Inner loop with i =", i, "and j =", j)
    print()
```

Notice here that since the inner loop is part of the body of the outer loop, it is executed three times. (So the body of the inner loop is executed in total $3 \times 4 = 12$ times). Note that it is essential to choose different names for the loop variables of nested loops.

Try now entering the following function definition that displays a "numbered" triangle:

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

and invoke it with, for example:

```
triangle(4)
```

Try to fully understand how this function works. In particular, make sure you can answer the following questions: Which loop provides the "numbers" for the lines of the triangle? Which loop displays each line of text in the triangle? What part of the code ensures that the lines are of different lengths? What role does the `end=" "` play after the first `print` statement? What role does the second `print` play, and is this statement part of the inner or

the outer loop?

Programming exercises

Begin by downloading the file pract06.py from Moodle to your filespace. Your solutions to the practical exercises should be added to this file, and this should be made available (electronically) in next week's practical session. Solutions to all of the exercises should use `if` statements, `for` loops or both. Try to ensure that your use of these structures is not only correct, but is as simple as possible.

1. A fast-food company charges £1.50 for delivery to your home, except for large orders of £10 or more where delivery is free. Write a function `fastFoodOrderPrice` that asks the user for the basic price of an order and prints a "The total price of your order is ..." message containing the total price including any delivery charges.

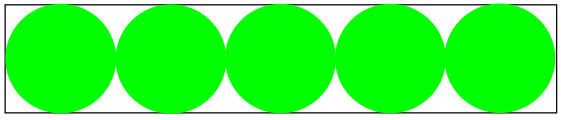
2. Write a `whatToDoToday` function that asks the user to enter today's temperature, and then prints a message suggesting what to do. For temperatures of above 25 degrees, a swim in the sea should be suggested; for temperatures between 10 and 25 degrees (inclusive), shopping in Gunwharf Quays is a good idea, and for temperatures of below 10 degrees it's best to watch a film at home.

3. Write a function `displaySquareRoots` that has two parameters, `start` and `end`, and displays the square roots of numbers between these two values, shown to three decimal places. For example, the call `displaySquareRoots(2, 4)` should result in the following output:

```
The square root of 2 is 1.414
The square root of 3 is 1.732
The square root of 4 is 2.000
```

4. A school teacher marks her pupils' coursework out of 20, but needs to translate these marks to a grade of A, B, C or F (where marks of 16 or above get an A, marks between 12 and 15 result in a B, marks between 8 and 11 give a C, and marks below 8 get an F). Write a `calculateGrade` function that takes a mark as a parameter and ***returns*** the corresponding grade as a single-letter string. If the parameter value is too big or too small, the function should return a mark of X.

5. Write a function `peasInAPod` that asks the user for a number, and then draws that number of "peas" (green circles of radius 50) in a "pod" (graphics window of exactly the right size). E.g., if the user enters 5, a graphics window of size 500×100 should appear like:



6. A train company prices tickets based on journey distance: tickets cost £3 plus 15p for each kilometre (e.g. a ticket for a 100 kilometre journey costs £18). However, senior citizens (people who are 60 or over) and children (15 or under) get a 40% discount. Write a `ticketPrice` function that takes the journey `distance` and passenger `age` as parameters (both integers), and ***returns*** the price of the ticket in pounds (i.e. a `float`).

7. Write a `numberedSquare` function that has a parameter `n` and displays a "numbered" square of size `n`; e.g, a call `numberedSquare(4)` should result in the output:

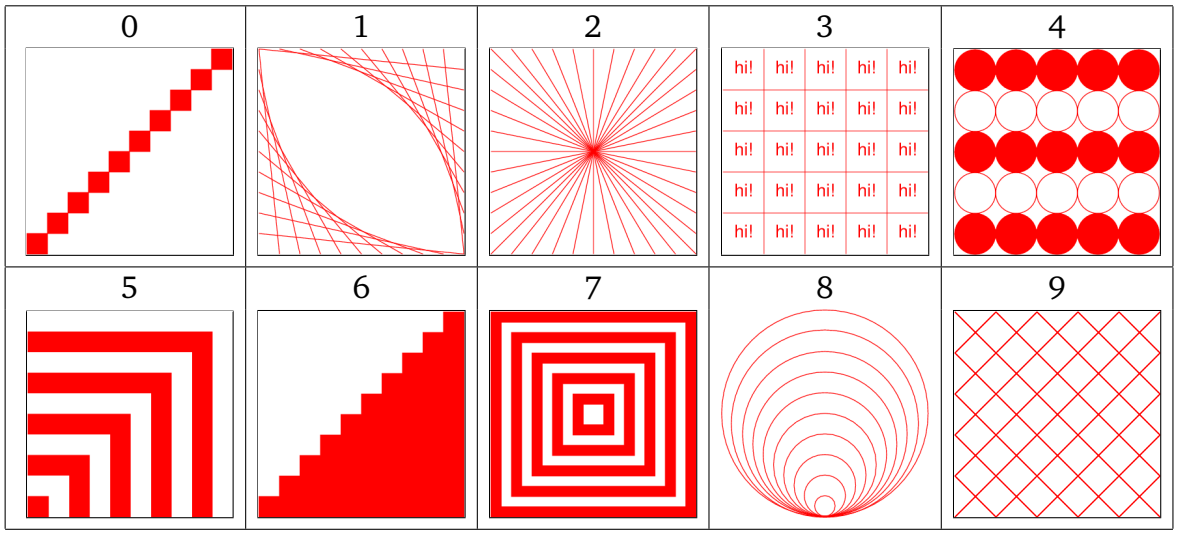
```
4 5 6 7
3 4 5 6
2 3 4 5
1 2 3 4
```

   (Notice that the top-left figure should should always be `n`.)

8. The pract06.py file contains an incomplete `drawColouredEye` function for which the graphics window, centre point, radius and eye colour (a string) are given as parameters. Complete the `drawColouredEye` function so that it draws an eye like those from last week, but for any given colour. Write another function `eyePicker` that asks the user for the coordinates of the centre of an eye, its radius and colour. If the user inputs a valid eye colour (blue, grey, green or brown), then an appropriately coloured eye should be displayed in a graphics window. Otherwise, just a "not a valid eye colour" message should be output. (Note: remember to avoid repetitive code.)

9. This exercise and the following one help to prepare you for the Python assignment.

   The table below shows 10 different "patch designs". Each patch design features a regular arrangement of lines, circles, rectangles and/or text, and has dimensions of 100×100 pixels.

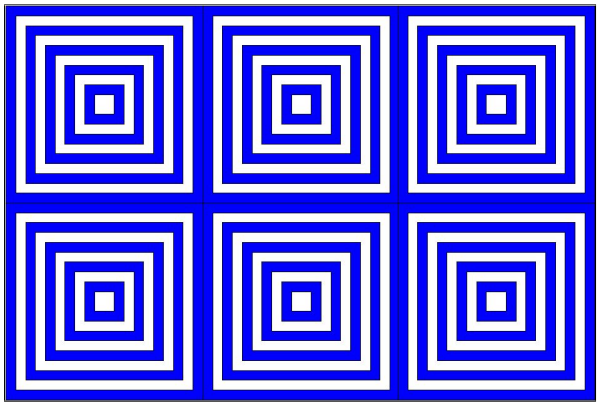


   Write a function `drawPatchWindow` (without parameters) which displays, in a graphics window of size 100×100 pixels, the patch design which is labelled with the ***final digit*** of your student number. The patch design should be displayed in red, as in the table. It's important that your program draws the patch design accurately, but don't worry if one pixel is chopped off at the edge of the window. Note that you don't need to draw a border around the patch design.

10. Write a function `drawPatch` which draws the same patch design, but which takes four parameters: the window in which to draw the patch, the x and y coordinates of where the top-left corner of the patch should be, and the colour of the patch.

    Write another function `drawPatchwork` (without parameters) which uses `drawPatch` to draw a blue "patchwork" three patches wide and two patches high in a graphics

window of size 300×200 pixels. For example, if your student number ends in 7, the following patchwork would be displayed:



11. Write an `eyesAllAround` function that allows the user to plot exactly 30 eyes on a graphics window of dimensions 500 by 500 by clicking on chosen centre points. All eyes should be of radius 30, but they should be of different colours: specifically, the colours should repeatedly cycle through the sequence "blue", "grey", "green" and "brown". Your function should call `drawColouredEye` to draw each eye.

12. [harder] Write an `archeryGame` function. This function should draw a target (like that from worksheet P03) using the supplied `drawCircle` function. Your function should then allow the user to click on the graphics window five times, representing the firing of five arrows – each click representing the point on the target that is aimed at.

    Fluctuating ***atmospheric conditions*** should be considered – your solution should generate two random values representing the amount an arrow will move horizontally and vertically from the aimed position during its flight, and these values should be adjusted a little (again randomly) after each arrow.

    Generating random numbers is easily accomplished using the `randint` function from the `random` module. This function takes two arguments and returns a random number between these two values. For example, try:

    ```
    import random
    for i in range(10):
        print(random.randint(1, 5))
    ```

    to give ten numbers randomly chosen between 1 and 5.

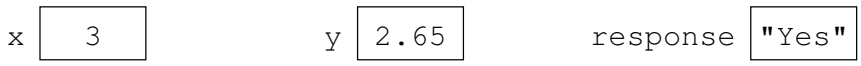
    Each time the user clicks, the function should (i) display a small black circle representing where the arrow hits, and (ii) record the number of points scored. The points awarded for each arrow (click) are as follows: 10 for the yellow area, 5 for red and 2 for blue. After the final arrow, the function should display the total score on the graphics window. (Hint: calculate distances by first importing your pract05.py file and using the `distanceBetweenPoints` function.

Written exercises

The aim of these exercises is to help you understand how simple Boolean expressions and `if` statements work.

Simple Boolean expressions

Suppose that there exist three variables `x`, `y` and `response` with the following values:



Evaluate each of the following Boolean expressions to give `True` or `False`:

1. `x == 3`
2. `x < 3`
3. `x <= 3`
4. `x != y`
5. `x >= y`
6. `x == int(y)`
7. `x == round(y)`
8. `response == "yes"`
9. `response[0].lower() == "y"`
10. `"No" < response`
11. `type(x) == type(y)`
12. `type(x) == type(0)`

If statements

Consider the following `if` statement, which contains other (nested) `if` statements:

```
a = eval(input("Enter a value: "))
b = eval(input("Enter a value: "))
c = eval(input("Enter a value: "))
if a > b:
    if b > c:
        print("Spam")
    else:
        print("Jam")
elif b > c:
    print("Ham")
    if a >= c:
        print("Sandwich")
    elif b == 5:
```

```
        print("Omelette")
else:
    if a == b:
        print("Pepperoni")
    else:
        print("Marmalade")
    print("Pizza")
```

What is the output from this code for the following input values?

13. 5, 4, 1
14. 5, 4, 7
15. 3, 5, 2
16. 1, 5, 2
17. 3, 3, 3
18. 1, 4, 2
19. 3, 4, 5

Simplifying if statements

20. Suppose that a program includes a Boolean variable called `isRight`. The following piece of program code is unnecessarily complex. Simplify it.

```
if isRight == True:
    print("Oh, yes it is!")
else:
    print("Oh, no it's not!")
```