Practical Sheet 3

If Statements

Aims

Section		Aim
1	Introduction	Introduce a running example to be developed in stages
2	Using Random Numbers	Understand how to get random numbers in a program
3	Simple If Statements	Understand how to use an if statement to choose between two possibilities
4	Conditional Expressions	Learn to use expressions that can be true or false
5	If and Else Statements	Understand more complex forms of choice
6	Enhancements	Add more enhancements to the program.

Related topics

- ***Topic 3.1 If Statements***
- ***Topic 3.2 Dry Run***

1 Introduction

A Simple Quiz Here is an example of running a simple quiz program (with user's input highlighted in red):

```
>>> 
Guess a secret number between 0 and 9
Go on! Guess it> 5
The correct answer was 9
>>> ================================ RESTART ==================
>>> 
Guess a secret number between 0 and 9
Go on! Guess it> 5
Good guess. Well done!
>>>
```

In this session we will write this program (and improve it). There are the steps:

Random!	Introduces random numbers, useful in the quiz
What? If	Explains how a program's statements can vary
Better Conditions	Conditions that can be used to vary the statements
If … Or Else	More complex conditional forms
You're got problems	Enhance the program to do more

2 Using Random Numbers

Exercise 2.1 Create a new file, copy and try the following program:

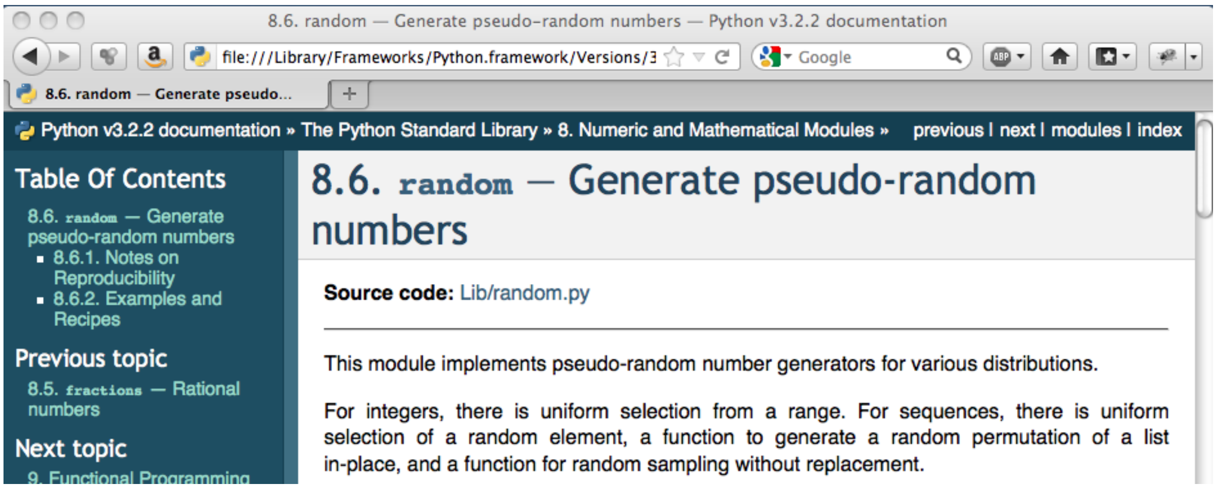
```
import random                 # Imports a 'module'
r1 = random.randrange(0, 10) # random number from 0 (included)
                              # to 10 (not included)
print(r1)
r1 = random.randrange(0, 10)
print(r1)
```

The function 'randrange' generates a random number. You should see two different numbers printed out, even though the function call is the same (there is a 10% chance that the numbers will be the same). Try running the program several times.

Modules

The first line 'imports' a module. This says 'use this module in this program'. A module contains *code written by some else that you can use*. There are lots of modules in Python but there is no need to remember all the details: *read the documentation*.

Part of the documentation for the 'random' module in the Python Standard Library. You can look at this from the help menu in your Python system.


8.6. random — Generate pseudo-random numbers — Python v3.2.2 documentation
file:///Library/Frameworks/Python.framework/Versions/3
Google
8.6. random — Generate pseudo...
Python v3.2.2 documentation » The Python Standard Library » 8. Numeric and Mathematical Modules »
previous | next | modules | index
Table Of Contents
8.6. random — Generate pseudo-random numbers
8.6.1. Notes on Reproducibility
8.6.2. Examples and Recipes
Previous topic
8.5. fractions — Rational numbers
Next topic
9. Functional Programming

8.6. random — Generate pseudo-random numbers

Source code: Lib/random.py

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

3 Simple If Statement

Here is a simple program:

```
import random

number = random.randrange(0, 10)
print ("Guess a secret number between 0 and 9")
guess = int(input("Go on! Guess it>"))

if guess == number :
    print("Good guess. Well done!")
```

This program introduces an 'if' statement:

```
if condition :
    statement that is executed only if condition is true
```

Exercise 3.1 Copy this program and try running it a few times. From time to time you guess correctly and the program print a cheery message, but if you don't the program does nothing. The next two steps fix this.

Indentation – It Matters!

Did you notice that the '`print("good guess` … is indented. The indentation shows that the statement is inside the if-statement. **Be careful** to layout the program neatly – it may mean something different if the indentation varies.

4 Conditional Expressions

In step two, we checked if the guess was equal to the secret number. We can write other conditions:

```
guess == number     # guess is equal to number
guess > number      # guess is more than number
guess < number      # guess is less than number
```

Exercise 4.1 Use these conditions to enhance the program to have three possible behaviours

```
Guess a secret number between 0 and 9
What's your guess>4
Bad luck: too low. The correct answer was 6
>>>
Guess a secret number between 0 and 9
What's your guess>7
Bad luck: too high. The correct answer was 3
>>>
Guess a secret number between 0 and 9
What's your guess>5
Good guess. Well done!
```

5 If and Else Statements

One solution to the program in the last section is:

```
import random

number = random.randrange(0, 10)
print ("Guess a secret number between 0 and 9")
guess = int(input("What's your guess>"))

if (guess == number):
    print("Good guess. Well done!")

if (guess > number):
    print("Bad luck: too high.")
    print("The correct answer was", number)

if (guess < number):
    print("Bad luck: too low.")
    print("The correct answer was", number)
```

In this program, there are there 'if' statements. Here's what the program does:

1. It checks if the numbers are equal, printing 'well done' if they are.
2. It checks if the guess is too high.
3. It checks if the guess is too low.

It always does **all three** checks, even though **only one** ever succeeds. For example, if the guess is correct then the two following checks are pointless. For situations like this, Python allows us to write a more complex if statement:

```
if condition :
    statement that is executed only if condition is true
else :
    statement that is executed only if condition is NOT true
```

and an even more general form:

```
if condition1 :
    statement that is executed only if condition1 is true
elif condition2 :
    statement that is executed only if condition2 is true
else :
    statement that is executed only if BOTH conditions are NOT true
```

Exercise 5.1 Use this to rewrite the program shown at the start of the step. Rewrite it in as many different ways as you can. Check with other people to see if they have alternative.

Which one do you prefer?

Equivalent Programs and Programming Style – Does It Matter?

Which of the many possible versions of the program is best? Programmers call this 'style'. You may aim for 'clarity' or 'cleverness'. I prefer the style that makes errors least likely.

The idea of many possible programs for one problem is very important:

- Understanding that two programs are the same (equivalent) is a way to understand programs. As a comparison, explain the importance of understanding that 3 x 4 is the same as 4 x 3.
- Programs that produce the same output (i.e. work the same) can still be different. One can be shorter but harder to understand; another can be longer. In particular, you cannot mark a program just by comparing it to a sample answer.

6 Enhancements

Here are some suggestions for improving the program:

- Give a different response if the guess is close to the correct answer. (E.g. "That was close. Bad luck!")
- Guess a 2 (or 3 digit number) and give prizes (1st, 2nd, 3rd ...) if close guess.
- Ask how many digits the secret number should have.
- Make the program to guess a random letter instead. (Hint: there isn't a library method for a random letter but suppose you had a string "ABCDEF..." and a random number between 0 and 25 inclusive.)

7 Summary Points

- If statements allow the statements in the program to vary
- If statements use conditions.
  - A condition can test for equality
  - A condition can test greater or less than
  - Python allows other condition: have a look at the documentation.
- An 'if' statement can have an else part.
- There are often many ways to write a program. The different solutions may behave the same way but differ in others ways, such as clarity.