

PYTHON

Introduction to Structured Programming

userweb.port.ac.uk/~poolem/python

Practical Worksheet 1: Getting started with Python

(This worksheet does not form part of the unit assessment)

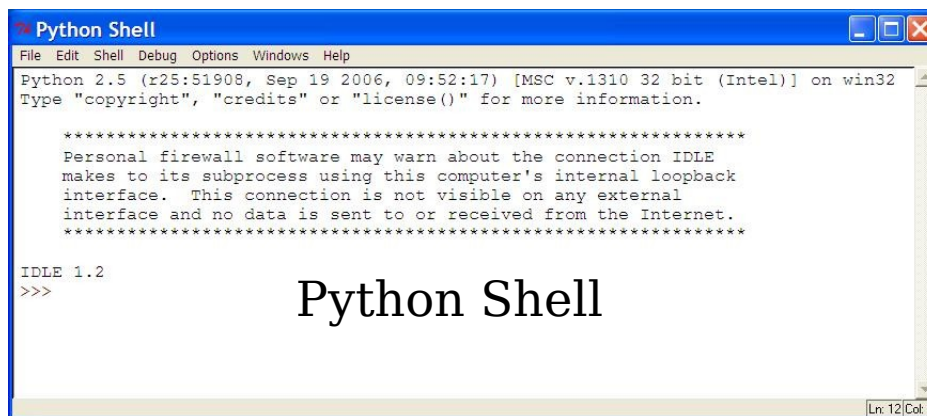
Introduction

This worksheet is designed to get you started with the Python language and the Python programming environment, Idle. Work through it carefully at your own pace, making sure you understand each part before moving on to the next. In particular, when the worksheet asks you to type something in, first try to predict what it will do, type it in, and finally make sure you've understood what has happened.

You are not expected to finish worksheets in your practical session, but you should complete as much as you can before the practical of the following week - it is vital that you spend a few hours of your own time each week developing your programming skills. Use the practical session to ask questions on those exercises which you are finding difficult. If you are having major difficulties understanding the concepts, then recommended sources of help outside the lectures and practical sessions are: your fellow student programmers, me, the unit textbook (Zelle), and the Tutor Centre (Lion Gate 0.04).

Start IDLE

The programming environment that comes with the Python software is called Idle. To start Idle, select General Applications from the Start menu, then Development Tools, Python 2.5, Idle (Python GUI). A window similar to the following will appear:



This window is called the **Python Shell**, or just Shell. It interprets (executes) Python code as you type it in, line-by-line, giving immediate feedback. This provides a good way to learn the basics of the Python language and to test out new concepts and ideas.

Your first Python program

At the shell prompt `>>>`, type the following:

```
print "hello, world!"
```

(followed by return). Idle executes the entered line, and once finished re-displays the `>>>` prompt ready for you to enter more code.

Note that each line of Python code is called a **statement**, and this particular statement is known as a **print statement**. This statement can also be considered to be a complete, albeit very simple, Python program!

Experimenting with the Python language

Let's continue using the prompt to experiment with the basics of the Python language. We have already seen above that we can use `print` to display text on the screen; `print` can also be used to display values of **arithmetic expressions**. Try, for example:

```
print 7
print 12 + 27
print 5 + 4 * 3
```

In fact, the shell will display the value of any expression that you type directly in at the prompt without using `print`; for example, enter:

```
5 + 4 * 3
```

The shell can thus be used as a simple calculator. Another useful feature of the shell is that you can re-display previously executed statements by holding down the Alt key and pressing p (for previous) or n (next). Once displayed, these statements can be modified and/or re-executed.

Variables

Variables are basic elements of any program (in any programming language). Each variable has a name, and refers to a location in the computer's memory that holds a value. The value of a variable may change during execution of a program.

Let's experiment with some variables. When you start the shell there are no variables at all. To create a new variable, we use an **assignment statement**; type:

```
x = 7
```

All assignment statements have this basic form: on the left of the = is a variable (here x), and on the right hand side is an expression (here 7). This assignment statement means: “create a new variable x, and assign it the value 7”.

To check the value of your new variable, type:

```
print x
```

or simply:

```
x
```

Let’s create another variable y, using an assignment statement with a slightly more complicated expression:

```
y = x + 1
```

This assignment statement says: “create a new variable y, and assign it a value one greater than the current value of x”. After entering this assignment, display the value of y.

Values of existing variables are changed using assignment statements. Try these statements, and inspect the values of x and y after each one:

```
x = x * 2  
y = x + y
```

Experiment with variables and assignment statements until you are confident that you understand their behaviour. (By the way, a variable doesn’t have to have a short name like x or y – just about any mix of letters and digits is fine, as long as it starts with a letter.)

Errors

Like all programming languages, Python has very strict rules in terms of what you can ask it to interpret, and will report an error if you break these rules. There are two types of error that Python will detect:

Syntax errors are mistakes in the form (structure) of the code. For example, try the following:

```
print "hello world"      (misspelt print)  
print hello world"      (missing quote)  
PRINT "hello world"     (print not in lowercase)  
x = y + * 4              (badly formed expression)
```

Semantic errors are mistakes in the meaning of the code. Try, for example:

```
print 2 * crash  
x = bang  
y = wallop + 3
```

These statements are syntactically OK (they look like valid print and assignment statements), but they don't make sense (assuming the variables crash, bang and wallop don't exist). After all, if a variable doesn't exist, it cannot have a value!

Writing longer programs using the editor

Whilst the shell is great for experimenting with Python, it is only practical for short programs, and when we close the shell, we lose any program we've typed in! We need to be able to write longer programs, and to save them as files on the disk so that they can be executed over and over again. In order to do this, we will use Idle's **editor**. To start the editor, select New Window from the File menu – a window like the following will appear.



Begin by typing a copy of the first (hello world) program into the editor. Save it to disk using Save As from the File menu. Save it with filename hello.py into a suitable folder (I recommend creating a Python folder and storing all your Python files there). Make sure you always type in the .py suffix when saving your programs, so they can be recognised as Python programs by Windows, and to ensure that they are displayed properly (with coloured text) within Idle.

Execute the program by selecting Run Module from the Run menu (or by pressing F5). Python executes the program within the shell window. The program may be executed as many times as you like by pressing F5 again.

Now change the program so that it looks exactly as follows:

```
def sayHello():  
    print "Hello world"
```

Execute the program again by pressing F5.

This program will not give any output in the shell. Instead, it defines a new **function**. A function (or **subprogram**) is a piece of code that has a name that we can use later. This particular function has the name sayHello. Now, try typing:

```
sayHello()
```

in the shell. Try it again. You should be able to see what is happening here – this `sayHello()` statement is a **function call**, it uses (or “calls”) the `sayHello` function we have just defined.

The Kilos to Pounds program

Now, download the kilos to pounds program `kilos2pounds.py` from the unit web-site. (Go to userweb.port.ac.uk/~poolem/python, click on the `kilos2pounds.py` link and save the file to your Python folder.) Open the file from the Idle editor – as you can see, it's the program from Lecture 1. Execute this program using Run Module or F5. As before, the program is executed in the shell; this time however, the program includes code that asks you to supply some **input**. Enter a number, and the program will give you the output before ending.

Let's make the `kilos2pounds` program a little bit more complete by adding a few **comment** lines at the top of the program:

```
# A kilograms to pounds conversion program
# Your name, 123456 (your number)
# 5th October 2009
```

(using your name and student number in the second line). Python will ignore any text that appears to the right of a `#` symbol, so this is how we **document** a program to make it more readable to yourself and fellow programmers. Execute the program again to see that the comment lines haven't changed what the program does.

Looking ahead: lists and loops

To make things a little more interesting, let's take a sneak preview of some of the programming concepts that we'll study in depth later in the unit.

First, using the shell, enter the following statement at the shell prompt:

```
print range(10)
```

We see that `range` gives us a **list** of numbers that begins with 0, and ends one short of the specified number.

Now try:

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

(To indent the second line, you may have to press `tab` once before the `print` – this will insert the correct number of spaces.) You'll need to press `return` twice after the second line to tell the shell that you've finished. This

statement is known as a **loop**. Loops are used in programming to execute a piece of code over and over again. Try to guess what is happening when this code is being executed. Note that the `i` is just a variable that is created and used by the loop.

Program files with many functions

Now, download the `week01.py` program from the web-site and save it into your Python folder. Open this program in the editor window. As you can see, this program contains three function definitions: `sayHello` (as seen already), `count` and `kilos2pounds`. The `count` and `kilos2pounds` functions contain code you've seen before. The blank lines between the function definitions are there just to make the code easier to read.

Execute this program in the normal way (press F5). Note that the program gives no output. Instead, it has defined three functions, which can now be used (called) from the shell. Try typing the following at the `>>>` prompt:

```
sayHello()  
count()  
kilos2pounds()
```

This is how you'll write and execute our code for the next few practical worksheets. Each week you will construct a single program file containing several function definitions, and you'll test this code by calling the functions from the shell.

Programming exercises

Each week's practical worksheet will include a set of programming exercises (some of which may be selected from the unit textbook). In future worksheets, these exercises will form a part of the assessment of the unit, but the first four worksheets are unassessed.

It is important that you keep up-to-date with the practical exercises. As already mentioned, work at your own pace – you are not expected to complete the worksheet during a single practical session. You should, however, attempt to complete as much as you can in your own time before the following week's practical.

Each practical worksheet will begin with comparatively simple exercises, and will get more difficult towards the end. (Don't worry if you cannot do the questions marked [harder]. These are mainly intended for those students who would like a further challenge.) Remember that learning to program is fundamentally about developing your problem solving skills – the exercises are thus designed to make you think, and may take some time to complete.

After each exercise, remember to re-execute the program from the editor before testing the corresponding function in the shell. Begin by modifying the comments at the top of the week01.py file to include your name and student (six digit) number.

1. Write a function called `sayName` that displays your name.
2. Write a `sayHello2` function that uses two print statements to display the text:

```
    Hello
    World
```

3. Write a function `euros2pounds` which converts an amount in Euros entered by the user to a corresponding amount in Pounds. Assume that the exchange rate is 1.10 Euros to the Pound. [Hint: be sure to test your solution carefully.]
4. Write an `addUp` function that asks the user to enter two numbers, and outputs their sum.
5. Write a `changeCounter` function. This should ask the user how many 1p, 2p and 5p coins they have (using three separate questions), and then display the total amount of money in pence.
6. Write a `tenHellos` function that uses a loop to display "hello, world!" ten times (on separate lines).
7. Change the `count` function so that instead of counting from 0 to 9, it counts up in tens from 0 to 90.
8. [harder] Write a function `weightsTable` that outputs a two-column table of kilogram weights and their pound equivalents for kilogram values 0, 10, 20 ... 100. (Don't worry too much about formatting the table neatly.)
9. [harder] Write a `futureValue` function that uses a loop to calculate the future value of an investment amount, assuming an annual interest rate of 5.5%. The function should ask the user for the initial amount and the number of years that it is to be invested, and should output the final value of the investment using compound interest with the interest compounded every year.