Practical Sheet 8: Functions in Python

Aims

Section		Aim
1	Simple Functions	Understand the idea of a function: distinguish between defining and calling a function
2	Functions with Parameters	See that the behaviour of a function can vary depending on the parameters that are passed to it.
3	Functions with Global Variables	Understand the difference between using parameters and global variables in functions.
4	Further functions exercises	Practice using functions
5	Introduction to turtle graphics	Use Python's turtle graphics and consider its role in teaching Python

Related topics

Topic 8.1 Functions
Topic 8.2 Pseudo Code and Flowcharts
Topic 8.3 Python Turtle Graphics

Python and Functions

A fundamental idea in programming is breaking a problem down into parts. In fact, this idea is fundamental to all complex systems. For example, it allows different companies to make different parts. (Exercise: lists as many different parts of an aeroplane as you can.)

A function is the most important idea in programming for breaking a program down into parts. All programs use function – for example 'print' is a function.

This sheet is about writing your own functions.

1 Simple functions

Here is a simple program that uses a function to input a number:

```
## Simple function to input a number

def inNum():
    numS = input("Enter a number> ")
    return int(numS)

num1 = inNum()
num2 = inNum()

print("Total is:", num1 + num2)
```

Exercise 1.1: Examine the program and run it. Answer the following questions:

- How many times is the function 'inNum' called when the program runs?
- What is the type of the value returned by the function 'inNum'?
- Which other functions are called in the program?

2 Functions with Parameters

The 'inNum' function is rather limited as has no parameters. Here is a simple (and rather pointless) program that uses functions with parameters:

```
# In this program two functions are defined.
def double(param):
    doubleP = param * 2
    return doubleP

def treble(param):
    trebleP = param * 3
    return trebleP

# This is a main program that uses the functions
print("Twice 12 is", double(12))
print("Thrice 12 is", treble(12))
```

Type this program into a file, save and run it. Be sure you understand how it works.

Exercise 2.1: try the following changes to the program. You should save each program with a new name:

1. Input a number, instead of always having 12.
2. Use the double function to double a number and then double it again.
3. Create a new function 'square' to multiply the number by itself.

3 Functions with Global Variables

Here is a second version of the double function.

```
## This version of double uses a global variable
def double() :
    global num
    num = num * 2

num = 17
double()
print(num)
```

Exercise 3.1: run the program and examine it. Answer / try the following questions / changes:

1. What does the keyword 'global' do?
2. What does the 'double' function return? What effect does it have?
3. Change the program to input a number N and print twice N. You should use the 'inNum' function from page 1.

4 Further Exercise

4.1 Useful Functions

Exercise 4.1: Write a function that prompts a user for a 'Yes' or 'No' answer, allowing variant spellings and continue until a choice is made.

Exercise 4.2: Write a function that takes a list of 'choice' strings as a parameter, displays the choices with a number and prompts the user to pick an option by entering a number. Return the number.

4.2 Shopping Again

Exercise 4.4: Write the shopping list program (see the Programming Challenges sheet) using functions.

You may wish to use the following outline and complete the three functions. Note that the outline already runs, but nothing much happens so you can test each stage.

```
def addItems(sList) :
    print("Trying to add an item to the list")

def buyItems(sList, pList) :
    print("Trying to take an item from the shopping list")
    print("to add to the list of purchases")

def printLists(sList, pList) :
    print("print the lists here")

shopping = []
purchases = []

while True :
    cmd = input("Command> ")
    if cmd == 'A':
        addItems(shopping)
    elif cmd == 'B' :
        buyItems(shopping, purchases)
    elif cmd == 'P' :
        printLists(shopping, purchases)
    else :
        print("The command are A (add), B (buy), P (print)")
```

Compare this version of the program to one that does not use functions. Which do you prefer and why?

5 Introduction to Turtle Graphics

Turtle graphics is implemented in lots of languages, notably Logo. The essential idea is to draw a picture by moving a 'turtle' around on a paper.

- The turtle is at an (x, y) location, and facing is some direction
- The turtle holds a pen: the pen has thickness and colour
- The turtle draws as it moves
- Shapes can be filled with any colour
- Text can be written and input of numbers and strings is possible.

There is a list of Turtle functions at the end of this sheet.

The picture drawing is animated – i.e. you watch the turtle moving. This is good for debugging. Animation slows everything down; you can switch it off and go faster.

Although turtle graphics is most associated with patterns, any picture can be drawn, where a picture is about colouring pixels. Complex animation (e.g. games) and graphical user interfaces (i.e. a file saving dialog) is not part of turtle graphics.

5.1 First Turtle Program

The simplest turtle programs do not require loops, if statements or even variables.

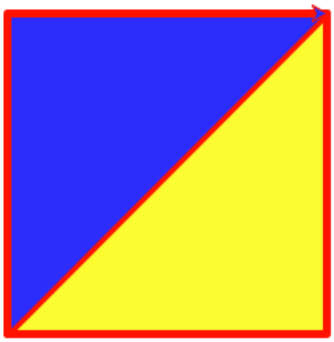
Exercise 5.1: Copy and run the following program.

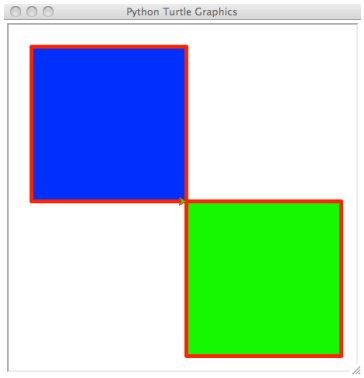
```
from turtle import *

pencolor('red')
pensize(5)

fillcolor('yellow')
begin_fill()  # start filling

forward(200)
left(90)
forward(200)
left(90)
forward(200)
left(90)
forward(200)
left(90)

end_fill()    # fill shapes drawn since start
done()  # the examples have this
        # but it seems to not be essential
```

Exercise 5.2: Adapt the program. Here are some suggestions

- Change the pen colour and fill colours
- Fill the square with two colours divided diagonally
- Draw two squares.

Exercise 5.3: Try drawing the following example pictures:





5.2 Using Functions in Turtle Programs

Because it takes quite a lot of commands to do anything, turtle graphics is good for teaching functions.

Exercise 5.1: Copy the following function:

```
def square(side):
    forward(side)
    left(90)
    forward(side)
    left(90)
    forward(side)
```

```
left(90)
forward(side)
left(90)
```

and call it in a program to draw 3 or more squares of different sizes.

Exercise 5.1: Try the following enhancements:

- Move the turtle using 'penup()' and 'goto()' so the picture has squares in different places.
- Enhance the square function to have a colour argument and fill the square with the given colour.
- Create a rectangle function.
- Create a similar function but instead of a square draw a regular polygon with a given number of sides, even of a given length.

6 Summary

Functions

- A functions is a reusable fragment of program, given a name that describes what it can do
- Simple functions do not use parameters, but parameters allow the data processed by a function to change. Most functions have parameters and a return value.
- A function can also access a 'global' variable. Global variables can sometimes be used instead of parameters but generally parameters are simpler.
- Functions are useful to break a complex program down into simple pieces. Describing useful functions is a good way to design a program.

Turtle graphics

- Turtle graphics can be used to introduce Python. Such an approach covers textual programming and problem solving well, but postpones coverage of variables.
- A turtle is moved by calling functions. Some 30+ functions are provided as part of the 'turtle' package.
- More complex patterns can be created by defining functions.

7 Appendix: Turtle Graphics Function Reference

The Python turtle graphics package has both

- A functional interface
- An object-oriented interface

We will use the functional interface; a few capabilities are not available as a result. The full documentation is found in chapter 23 of the Python standard library.

Function	Description and Example
Move and Draw	
forward(), backward()	Move the turtle a distance: `forward(10)`
right(), left()	Turn by an angle: `right(90)`
goto()	Move to an (x, y) position: `goto(10, 50)`
home()	Move to the home position: `home()`
circle()	Draw a circle or arc. Examples: • `circle(100)` – draw a circle of radius 100 • `circle(50, 90)` – draw an arc of radius 50, angle 90
dot()	Draw a dot with diameter & colour: `dot(20, 'blue')`
Turtle Position	
setheading()	Set the direction of the turtle: `setheading(90)`. In standard mode, 0 is East, 90 is North, 180 is West and 270 is South.
heading()	Get the heading: `angle = heading()`
xcor(), ycor()	Get the x or y coordinates: `currentX = xcor()`
distance()	Calculate the distance from the current position to some point: `dist = distance(50, 75)`
towards()	Calculate the angle from the current position to the given co-ordinates: `towards(100, 100)`
Pen and Turtle	
pendown(), penup()	Put the pen down / up. When the pen is down, moving the turtle draws a line.
pensize()	Set the pen size: `pensize(10)` for a thick pen.
isdown()	Returns true if the pen is up.
showturtle(), hideturtle()	Show or hide the turtle. It is faster to draw without the turtle visible. (*Note: it is also possible to switch off animation altogether*)
pencolor()	Set the pen colour: `pencolor('green')`

Function	Description and Example
Filling and Clearing	
fillcolor()	Set the fill colour; colours most easily entered as string though other formats supported: `fillcolor('blue')`.
filling()	Test whether shapes being drawn are to be filled.
begin_fill(), end_fill()	These command bracket drawing commands in which shapes should be filled.
reset()	Reset everything.
clear()	Clear the picture but do not reset the turtle.
write()	Write a text string: `write("Hello")` writes to the current position (which does not change) and aligns the string so that the turtle is at the left hand of the string.
Screen	
bgcolor()	Set the background colour of the screen: `bgcolor('pink')`
bgpic()	Set a picture as the background, using a file name.
screensize()	Set the screen size to e.g. 500 width and 300 high: `screensize(500, 300)`
textinput	Input text using a dialog: `textinput("Title", "Prompt")`
numinput	Input a number
window_height, window_width	Get the window height or width