Appendix 1

A Few Short Programs – written in Python

1) Print the first 10 Fibonacci numbers

Program:

```
x=1
y=1
for j in range(1,9):
    z=x+y
    print(z)
    x=y
    y=z
```

Notes:

a) for j in range(1,9):

This means

Put j=1 and perform the indented statements

Put j=2 and perform the indented statements

…

Put j=8 and perform the indented statements

Confusingly, for j in range(1,9) means j=1, 2, … 8 but not j=9

b) x=y

This means put x equal to the current value of y. So, for example

```
x=3
y=7
x=y
```

We now have x=7 and y=7

y=x This means put y equal to the current value of x. So, for example

```
x=3
```

y=7

y=x

We now have x=3 and y=3

So x=y and y=x do not mean the same thing!

c) We can run this program by keeping track of the values of j, z, x, y and see what gets printed.

j	z	print	x	y
			1	1
1	2	2	1	2
2	3	3	2	3
3	5	5	3	5
4	8	8	5	8
5	13	13	8	13
...				

This program will print the 2^{nd}, 3^{rd}, 4^{th}, … 10^{th} Fibonacci numbers which is almost what we wanted!

2) Use Euclid's algorithm to find the highest common factor of 3458 and 651

Program:

```
r=1
a=3458
b=651
while r!=0:
    r=a%b
    a=b
    b=r
print(a)
```

Notes:

a) while r!=0:

This means while r does not equal 0

Perform the indented statements

Perform the indented statements

… until r=0

Then continue with the rest of the program

When we start this while loop r has to have a value so we gave it a value in line 1

b) $r = a\%b$

This means put r equal to the remainder when a is divided by b So, for example

a=17

b=5

$r = a\%b$

We now have $r = 2$

c) We can run this program

r	a	b	print
1	3458	651	
203	651	203	
42	203	42	
35	42	35	
7	35	7	
0	7	0	7

d) We can change lines 2 and 3 to find the highest common factor of any two positive integers.

3) Test if 28 a perfect number

Program:

```
n=28
c=0
for j in range(1,n):
    if n%j==0:
        c=c+j
if c==n:
    print("perfect")
else:
    print("non-perfect")
```

Notes:

a) if n%j==0:

This means if the remainder when n is divided by j is zero

In other words, if j is a factor of n

We have to use == and not = in an if statement.

b) c=c+j

This means c adds up all the factors of n

c) if c==n:

This means if the sum of the factors of n (including 1 but excluding n) is equal to n

In other words, if n is a perfect number

4) Test if 17 is a prime number

Program:

```
n=17
prime=1
for j in range(2,n):
    if n%j==0:
        prime=0
        break
print(p)
```

Notes:

a) Once we have run the program, prime=1 if m is a prime number and prime=0 if m is not a prime number.

b) if n%j==0:

This means if j is a factor of n

In other words, if n is not a prime number.

c) break

This means jump out of the for loop and continue with the rest of the program

d) We can make this program part of a larger program to print out all the prime numbers less than, say 100

```
for n in range(3,100):
    prime=1
    for j in range(2,n):
        if n%j==0:
            prime=0
            break
    if prime==1:
        print(n)
```

5) Test if 21 is the sum of two squares

Program:

```
N=21
flag=0
for j in range(0,N+1):
    for k in range(j,N+1):
        if N==j*j+k*k:
            flag=1

            break
print(flag)
```

Notes:

a) We set flag=0 at the start. We only enact flag=1 if N is the sum of two squares.

b) print(flag)

This means we print 1 if N is the sum of two squares. Otherwise we print 0.

c) try running this program by hand.

6) Calculate $f(17)$ where $f(x)=2x+7$

Program:

```
def f(x):
    return (2*x)+7
y=f(17)
print(y)
```

Notes:

Lines 1 and 2 define the function f(x). We can put this at the start of the program and then refer to it later in the program.

7) Calculate u_{12} in the sequence $u_1=3 \quad u_{n+1}=2u_n+3n$

Program:

```
def u(n):
    if n==1:
        return 3
    else:
        return 2*u(n-1)+3(n-1)
t=u(12)
print(t)
```

Notes:

a) Line 3 tells us that $u(1)=3$

Line 5 tells us that $u(n)=2u(n-1)+3(n-1)$

$u_{n+1}=2u_n+3n$ and $u_n=2u_{n-1}+3(n-1)$ mean the same thing.

For example, if we put n=7 in the first version we get $u_8=2u_7+21$ and if we put n=8 in the second version we get $u_8=2u_7+21$

8) Calculate the 10^{th} derangement number using the recurrence relation

Remember $D_1=0$ and $D_2=1$ and $D_{n+2}=(n+1)D_n+(n+1)D_{n+1}$

Program:

```
def D(n):
    if n==1:
        return 0
    if n==2:
        return 1
    else:
        return (n-1)*D(n-2)+(n-1)*D(n-1)
h=D(10)
print(h)
```

9) Toss a coin 5 times and count the number of heads and tails

Program:

```
import random
h=0
t=0
for j in range(1,6):
    c=random.randint(0,1)
    if c==0:
        h=h+1
    else:
        t=t+1
print(h,t)
```

Notes:

a) import random

This means the program can generate random numbers

b) c=random.randint(0,1)

This means c is randomly set to 0 or 1.

Think of c=0 as tossing a coin and getting a head and c=1 as getting a tail.

10) Play the game Chuck – a – Luck a million times and find the average winnings per game

Program:

```
import random
T=0
for j in range(1,1000001):
    D=0
    dice1=random.randint(1,6)
    dice2=random.randint(1,6)
    dice3=random.randint(1,6)
    if dice1==6:
        D=D+1
    if dice2==6:
        D=D+1
    if dice3==6:
        D=D+1
    if D==0:
        W=-1
    else:
        W=D
    T=T+W
print(T/1000000)
```

Notes:

a) dice 1 is the score on the first dice, etc

b) D is the total number of sixes on the three dice in one game

c) W is how much I win in one game.

d) T is my total winnings in 1,000,000 games

e) If you work it out exactly, you will find that:

my average winnings, per game, in the long run, is $£-17/216$

Most(?) people are surprised by this as they would guess my average winnings, per game, in the long run would be positive. This, therefore gives you the opportunity, to extort money from people who have not read this book!

11) Estimate $\frac{\pi}{4}$ using random numbers see chapter: Pi CHECK

```
import random
N=1000000
T=0
for j in range(1,N+1):
    x=random.uniform(0,1)
    y=random.uniform(0,1)
    if x*x+y*y<1:
        T=T+1
print(T/N)
```

Notes:

a) N is the number of points we will generate in the unit square.

b) T is the number of these points that are inside the quarter circle.

c) x=random.uniform(0,1)

This means x is a random number between 0 and 1

d) x*x+y*y<1

This is true if the point (x, y) is inside the quarter circle.

NOTE

To run these programs, go online and find an online IDE. Choose Python as the language, type in the program and run it.

This is not a course in programming. These examples are here to entice you into learning some programming if you have not done any before. These programs could be improved. For example, in program (4) we do not need the range of the for loop to be (2,m) Why not?

Program (4) Test if 17 is a prime number. We can easily adapt this program to test any positive integer to see if it is a prime number. Or we could print out a list of the first 100 prime numbers. Or ...