Week 3 - Data Structurs and Recurrence

This lab sheet will introduce various data structures and also and important concept called 'recurrence'. After this session you will know how to:

- Create and manipulate lists;
- Create and manipulate dictionaries (hash tables);
- Write data to a file using the `write` and `read` functions;
- Use the csv python module to read and write csv files;
- Program some basic algorithms using recurrence.

Lists in Python

Lists are a particular object in Python that hold ordered collection of other objects. In other languages they are sometimes called 'arrays'. You can think of these as baskets that allow you to hold objects. You can put anything in lists:

- Numeric variables;
- Character variables;
- Other lists;
- and various other things.

1. **TICKABLE**: The following code creates a list with the numbers from 1, to 10.

   ```
   alist = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
   ```

   We can manipulate lists in a similar way to strings. Try the following:

   ```
   blist = [30, 40, 50, 60]
   clist = alist + blist
   print clist
   print len(clist)
   print clist[0]
   print clist[-1]
   print clist[3:12]
   ```

   We see that python indexes lists just like strings: starting at 0. We can also use the `index` function as for strings. Try:

```
index = clist.index(40)
print index
print clist[index:index + 2]
```

2. We have seen how to combine two list `+` but there is a very useful method on lists called the `append` method. With this we can easily add elements to lists:

```
mylist = []
for i in range(11):
    if i % 2 == 0:
        mylist.append(i)
print mylist
```

This makes use of the range function that we see in the previous lab sheet.

3. **TICKABLE** Create a list with the first 1300 integers divisible by 3.
4. There is another way of creating/manipulating lists in python called list *comprehensions*. The following code give the squares of the first 10 integers:

```
squares = [e ** 2 for e in range(1, 11)]
print squares
```

We can include logical statements to only give the squares of odd numbers:

```
squares = [e ** 2 for e in range(1, 11) if e %% 2 == 1]
print squares
```

5. **TICKABLE** By creating a function and using list comprehensions, create a list of $f(n)$ for all integers $n \leq 100$ where $f(n)$ is given below:

$$f(n) = \begin{cases} n^3, & \text{if } n \text{ odd} \\ n^2 + 1, & \text{if } n \text{ is divisible by } 4 \\ n - 1, & \text{otherwise} \end{cases}$$

Dictionaries in Python

6. **TICKABLE** In computer science 'hash tables' are used as an efficient way to find particular data that is used often. In python 'hash tables' are called dictionaries. To understand this consider the following list of lists:

```
badphonebook = [["Vince", 3], ["Zoe", 2], ["Julien", 6], ["Thomas", 10], ["Mike", 1], [
```

To find a particular phone number in this phone book we would need to go through ever element of the phone book to check if it was the right one:

```
def searchpb(target):
    for e in badphonebook:
        print "Checking %s" % e
        if e[0] == target:
            return e[1]
    return "%s not in phonebook" % target
```

Code this function and use it to find all the phone numbers in the above phone book. Try to find some strings that are not in the phone book.

In reality this is not how a phone book is designed. Names are in a given order (alphabetical) and so it is easier to know *where a name is supposed to be*. This is implemented in python using 'dictionaries' which are an **unordered set of *key:value* pairs**. This code creates the above phone book as a dictionary with the names as *keys* and the numbers as their *values*:

```
goodphonebook = {"Vince": 3, "Zoe": 2, "Julien": 6, "Thomas": 10, "Mike": 1, "Matt": 4}
```

To query a dictionary we can use the `get` method:

```
goodphonebook.get("Thomas")
goodphonebook.get("Brayden")
goodphonebook.get("Brayden", 'Not in book')
goodphonebook.get("Thomas", 'Not in book')
```

We can also modify an element of a dictionary as follows:

```
print goodphonebook['Vince']
goodphonebook['Vince'] = 8
print goodphonebook['Vince']
```

We must just be careful as if we use square brackets for a value that is not in a dictionary we will obtain an error:

```
print goodphonebook['Brayden']
goodphonebook['Brayden'] = 12
print goodphonebook['Brayden']
```

Note valid keys must be strings or numerical variables but anything can be a value of a key.

6. Iterate over the list `badphonebook` to initiate the `pb` as the equivalent dictionary:

```
pb = {}
for e in badphonebook:
    ...
```

7. Iterating over values in a dictionary. Note that it is also possible to iterate over keys in a dictionary:

```
for e in goodphonebook:
    print goodphonebook[e]
```

Writing data to files

8. **TICKABLE** All of the data we handle with variables, lists and dictionaries lives in the 'memory' of a computer when our python code is running. When the program stops running the data is lost. There will be occasions when we want to write our data to a file on the hard drive of a computer (so that it is always available even when we turn the computer off).

   To do this we need to open a file (usually a basic text file), write strings to the text file and then close the file. The following code opens (or creats a) text file in 'write mode' (that's what the `w` is for) and write the number 1 to 10 to it:

```
textfile = open('mytextfile.txt', 'w')
for i in range(1, 11):
    textfile.write("%s\n" % i)
textfile.close()
```

   Note that the string we are writing at each step of the loop ends with a `\n`. This is a special character that tells the writer to write a new line. There are other special characters such as `\t` which tells the writer to include a tabulated space.

9. To read data from a file, we need to open the file in 'read mode':

```
textfile = open('mytextfile.txt', 'r')
string = textfile.read()
print string
```

10. **TICKABLE** The following function checks if a number is prime or not. Read through the function and ensure that you understand it.

```
def isprime(n):
    return max([e % n for e in range(2, n)]) != 0
```

   In t

Recurrence

Programming in Python

Basic variables

A value is one of the basic building blocks used by a program. Values may be of various types.

3. **TICKABLE**: Experiment with the following code which creates variables (by assigning them a value) and checks what type they are using the `type` function.

   ```
   num1 = 23
   print type(num1)

   num2 = 23.5
   print type(num2)

   str1 = "Hello world!"
   print type(str1)
   ```

   Video hint

4. We can carry out basic arithmetic operations using python. Take a look at the following:

   ```
   num = 2
   num = num +3
   print num
   ```

   this can however also be written:

   ```
   num = 2
   num += 3
   print num
   ```

   Simlarly, `-`, `*`, `/` and `**` can be used for:

   - subtraction;
   - multiplication;
   - division;
   - exponentiation.

Video hint

5. **TICKABLE**: Assign the variable `num` to a value of 5.2, what is the result of adding 7 to `num` then muliplying `num` by 300 then dividing `num` by 4 and finally raising `num` to the power of 3?

   Video hint

6. **TICKABLE**: We can carry also manipulate strings. Try out the following:

   ```
   str1 = "This is a string that I will learn to manipulate"
   str2 = ", string manipulation is very useful."
   string = str1 + str2
   print string
   print len(string)
   print string[0]
   print string[-1]
   print string[3:7]
   ```

   We see that python indexes a string, starting at 0, we can also use negative values to start from the end.

   ```
   index = str1.index("string")
   print index
   print str1[index:index + len("string")]
   ```

   There are various other things that can be done "on" strings, be sure to research these.

   Video hint

7. It is possible to go from one type of variable to another.

   ```
   f = 10.2
   print int(f)
   print float(int(f))

   s = str(f)
   print s
   print type(s)
   ```

   It is also possible to write strings using other variables.

   ```
   numberofcats = 2
   name = "Vince"
   height = 1.7
   notborn = "the UK"
   ```

One way to do this would be:

```
string = "My name is " + name +", I am " + str(height) + " metres tall, have " + str(nu
```

Python (and most other languages) has a nicer way of doing this:

```
string = "My name is %s, I am %.2f metres tall, have %i cats and was not born in %s" %
```

The `%` is used to denote that a value must be input in to the string. The symbols after the `%` say what type of value is to be included:

- `s`: A String
- `.`x`f`: A float rounded to x decimal places
- `i`: An integer

There are other types that can be used as well.

Video hint

If statements

8. An `if` statement allows you to tell a program to carry out something based on the value of a `Boolean` variable.

```
boolean = True
if boolean:
    print "boolean is %s" % boolean
```

Try typying the above code but change `boolean` to `False`. **Note: in python, indentation is important! In all languages it is good practive, in python it is a requirement**.

It is easy to create boolean variables using the following:

- `<`: strictly less than
- `>`: strictly greater than
- `<=`: less than or equal
- `>=`: greater than or equal
- `!=`: not equal
- `==`: equals **Note: this is a test of equality as opposed to the basic `=` which is an assignment.**

It is also possible to give alternatives to an `if statement`:

```
num = 11
print num % 2 == 0
if num % 2 == 0:
    print "num is an even number"
else:
    print "num is an odd number"
```

(The `%` operator gives the remainder of one number when divided by another.)

Video hint

Spend some time understanding the `elif` statement.

9. **TICKABLE**: Find some information on the `raw_input` statement and write some code that prompts a user to input a string. If the length of that string is more than 10 then print “that string has length strictly more than 10” otherwise “that string has length less than 9”.

   Video hint

Loops

An important type of programming instruction allows us to make a program repeat certain things. These are also referred to as loops. There are two basic types of loops “count controlled loops” and “event controlled loops”.

10. The `range()` function in python allows us to create a list of integers easily. **A list is a new type of variable that we will look at more closely next week**:

    ```
    print range(10)
    ```

    Note that this gives a list starting at 0 of size 10 (so it goes up to the integer 9). We can include 2 arguments in to this function:

    ```
    print range(3,10)
    ```

    We can also include 3 arguments:

    ```
    print range(0,10,2)
    ```

    Using `range()` we can use the basic `for` loop in python (a type of count controlled loop):

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

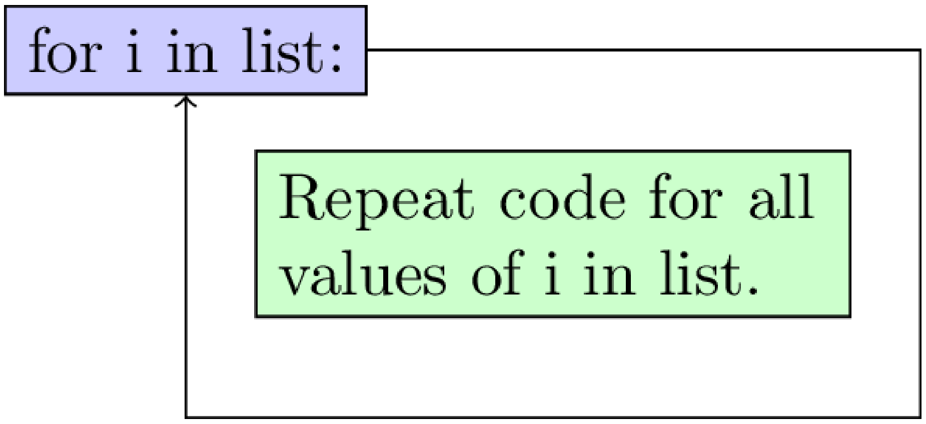

Figure 1: A basic For loop

The first line of the above defines the iterator `i` and tells it the values it will go through as shown in Figure 1.

We can in fact iterate over anything in a list:

```
for e in ["dog", "cat", 3, "I love mathematics"]:
    print e
```

This allows us to do various interesting things. Try the following:

```
s = 0
for i in range(1001):
    s += i
print s
```

Video hint

11. **TICKABLE**: Modify the above code so that it calculates the sum of the first integers less than 1000 that are not divisible by 3.

    Video hint

12. Event based loops are implemented in python using a `while` command that keep repeating a set of commands until a boolean variable is `False`.

```
k = 0
while k < 10:
    print k
    k += 1
```

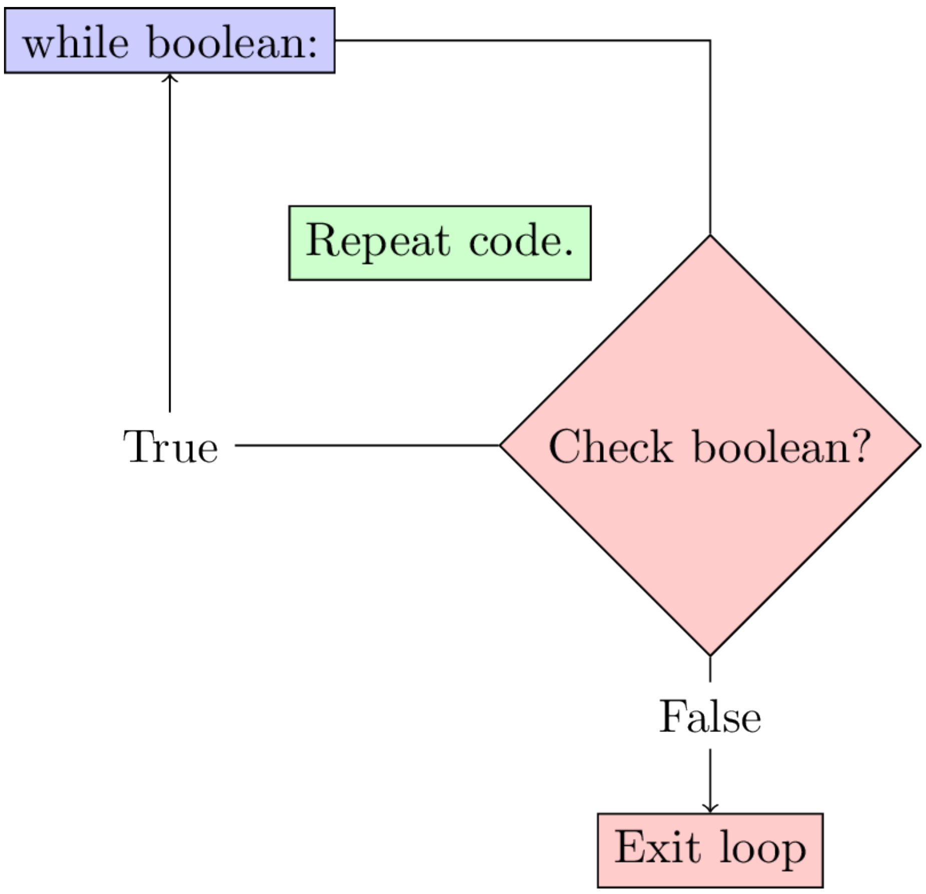

Figure 2: A basic while loop

The second line of the above checks the boolean variable `k < 10` and as long as this is `True` loops through the rest of the commands as shown in Figure 1:

Video hint

13. **TICKABLE**: Write some code to find N such that $\sum_{i=0}^{N} i^2$ is more than 20000.

    Video hint

14. It can be shown (you are not required to check this) that the following sequence:

    $$x_{n+1} = \frac{x_n + K/x_n}{2}$$

    approaches $\sqrt{K}$ as n increases. Write some code to verify this to any given level of precision.

    Video hint

15. Take a look at the `random` python library (we will talk about libraries in detail later) and write some code that uses the `input` function to code a simple game:

    - The program chooses a random integer;
    - The user tries to guess the integer;
    - At every guess the program indicates if the guess is too high or too low.

    Video hint

Functions

To be able to make progress from the basic on this sheet we need a way to write "recycle" code: functions. Much like mathematical functions, functions in programming can take multiple arguments and carry out tasks with those arguments as shown diagramaticaly in Figure 3.

16. The following code defines a very simple function (with no arguments):

    ```
    def printhello():
        print "Hello"
    ```

    The name of the function is `PrintHello` and `def` is the python syntax used to define it. When we run the above two lines of code, nothing is output. To call the function we simply write:

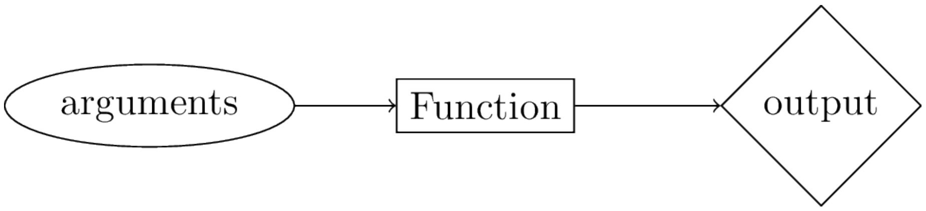

Figure 3: A diagram of a function

```
printhello()
```

We can modify our function to take an argument:

```
def printhello(name):
    print "Hello, " + name
```

Video hint

17. The following function makes use of the `return` call to actually return a result of the function:

```
def mydiv(a, b):
    return a/b
```

Video hint

18. **TICKABLE**: Include a check in the `MyDiv` function to ensure that no division by 0 is attempted.

Video hint

19. **TICKABLE**: Create a function that returns the sum of the first K integers not divisiable by B. Investigate "using optional arguments" and set K and B to have default values 10000 and 3 respectively.

Video hint

20. Create a function that return the square root of a number using the algorithm suggested in question 14. Write some code that compares the output of this algorithm to the actual square root for the first 10000 digits.

21. **TICKABLE**: Write a function `Fibonacci` that uses loops to calculate the nth number of the Fibonacci sequence:

$$X_n = \begin{cases} 1, & n = 0, 1 \\ X_{n-1} + X_{n-2} \end{cases}$$

Video hint

Writing clear code

When writing code it is **very important** to include comments throughout. How well commented code is will be evaluated throughout this module. Comments should be thought of as messages explaining what instructions are being given by the code. This is useful to the writer of the code but more importantly to anyone who might want to add/modify the code.

There are two ways of writing comments in python:

- Use the `#` to indicate to the interpreter that everything that is about to follow on a given line is to be ignored.
- Use `"""` to indicate the beginning and end of multi line comments (note that this can also be used to write multi line strings).

22. The following is an example of a single line comment in the middle of some code:

```
num = 2
num += 3  # Add 3 to num
print num
```

23. The following is an example of a multilined comment in the definition of a function:

```
def myfunc(a,b):
"""
This function calculates the ratio of two numbers raised to the sum of the two numbers.

Arguments:
    a: the first number
    b: the second number

Output: (a / b) ** (a + b)
"""
    return (a / float(b)) ** (a + b)
```

24. **TICKABLE**: One final aspect that is very important when writing code is **convention**. When working on a project with multiple people for example being able to use the same convention can be very beneficial. The most commonly known convention for python is PEP8. You are advised to use the following general summary of PEP8 for this course:

    - Variable and function names

Use a descriptive `lowercase` (all lowercase characters) for variable and function names.

Yes:

```
myvariable
sqrtvar
var
myfunction
```

No:

```
my_variable
SqrtVAR
MyFunction
MYFUNCTION
```

On some occasions it might be appropriate to make some exceptions (for example using a single letter for a very simple variable).

- White spaces

Include a whitespace between operators (`+`, `-`, etc) and a whitespace after a comma `,`.

Yes:

```
print 2 + 2
myfunc(3, 4)
```

No:

```
print 2+2
myfunc(3,4)
```

Include 2 whitspaces before an inline comment `#` at the end of a line of code.

Yes:

```
# Just leave a space after the comment symbol if on a single line
print 2 + 2  # but if you comment at the end of a line leave 2 whitespaces.
```

No:

```
print 2 + 2 # So this is not enough space.
```

Also include two blank lines before the definition of a function.

Yes:

```
print 2 + 2


def myfunc():
    print 2 + 2
```

No:

```
print 2 + 2

def myfunc():
    print 2 + 2
```

- Comments

Comment well and comment often. In particular use the following convention for functions:

```
afunc():
"""
Always start a function with a multiline comment to describe what it does.

Arguments: List the arguments and what format they should be in.

Output: List the expected output of the function.
"""
```

As and when we see new topics on this course we will also discuss the corresponding conventions.

Go back through your script and ensure that you have used the above convention.