

## Worksheet 7 – Planning

There are four major areas you must complete for your assessment program plan; a **flowchart**, a **variable table**, a **testing table**, and a **design sketch**. Let's recap the Level 2 content, then review the new elements we need to keep in mind when planning Level 3 programs.

### Flowcharts recap

One of the methods of planning is to use flowcharts which show a visual representation of how a program will work. They do not actually show the code that will be used for the program, only the logic.

There are strict flowcharting industry standards and conventions which dictate the shapes we use for the various logical elements of our flowcharts. Review the recap below for an explanation of these:

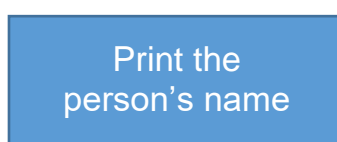
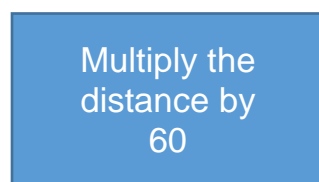
#### The start box

- All flowcharts must start somewhere - to start you use the start box (oval).
- They have the word START written in them – nothing else.
- You can colour the box if you want, but it must be an oval.



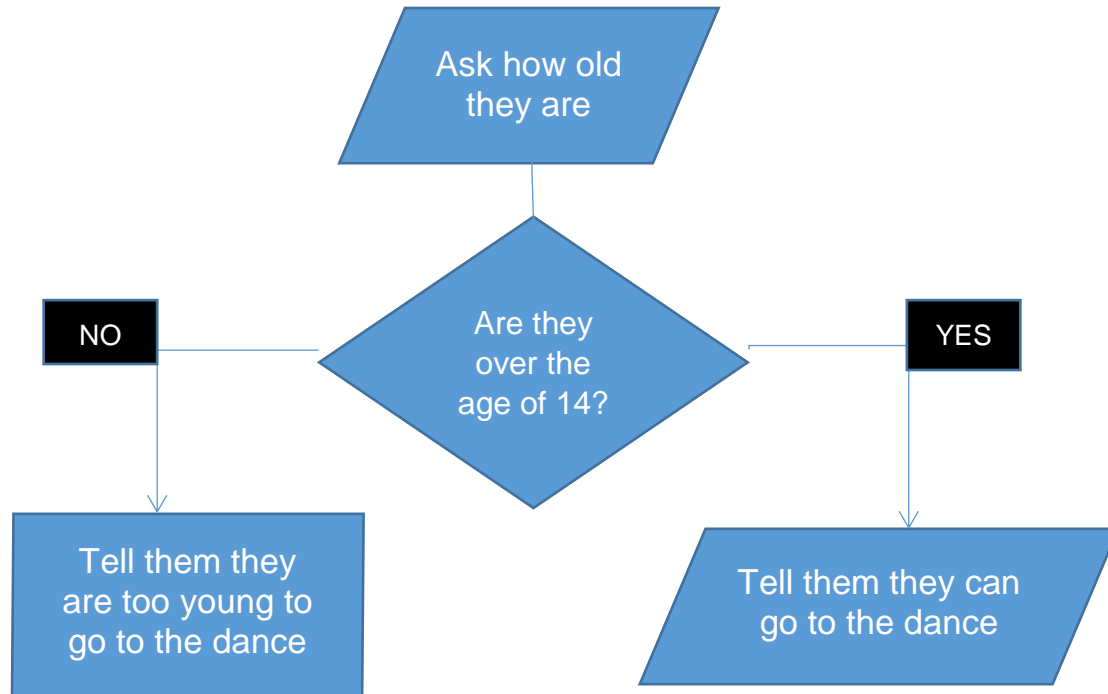
#### The process box

- Most every process that the program performs (except decisions – they use something else) that has interaction with the user or changes variables has a process box (rectangle).
- There is no need to have a process box to define variables – they are not included on the flowchart and are accounted for by creating a separate variable table.
- Real code is not used in flowcharts – remember that the flowchart shows the logic of the program, not the code that will be written to create it.

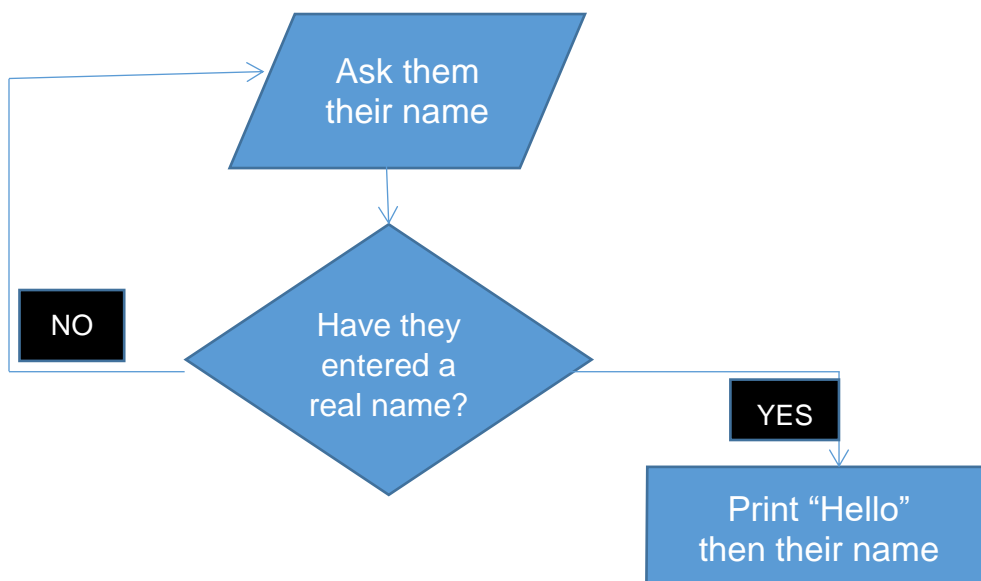


## The input box and decision box

- The **input box** (parallelogram) is used whenever input is needed by the program. How the input is gained is not important at this stage – it could be direct input, an input box, yes/no, etc.
- The **decision box** is used when the program can go in more than one direction. All decisions must be YES or NO questions.



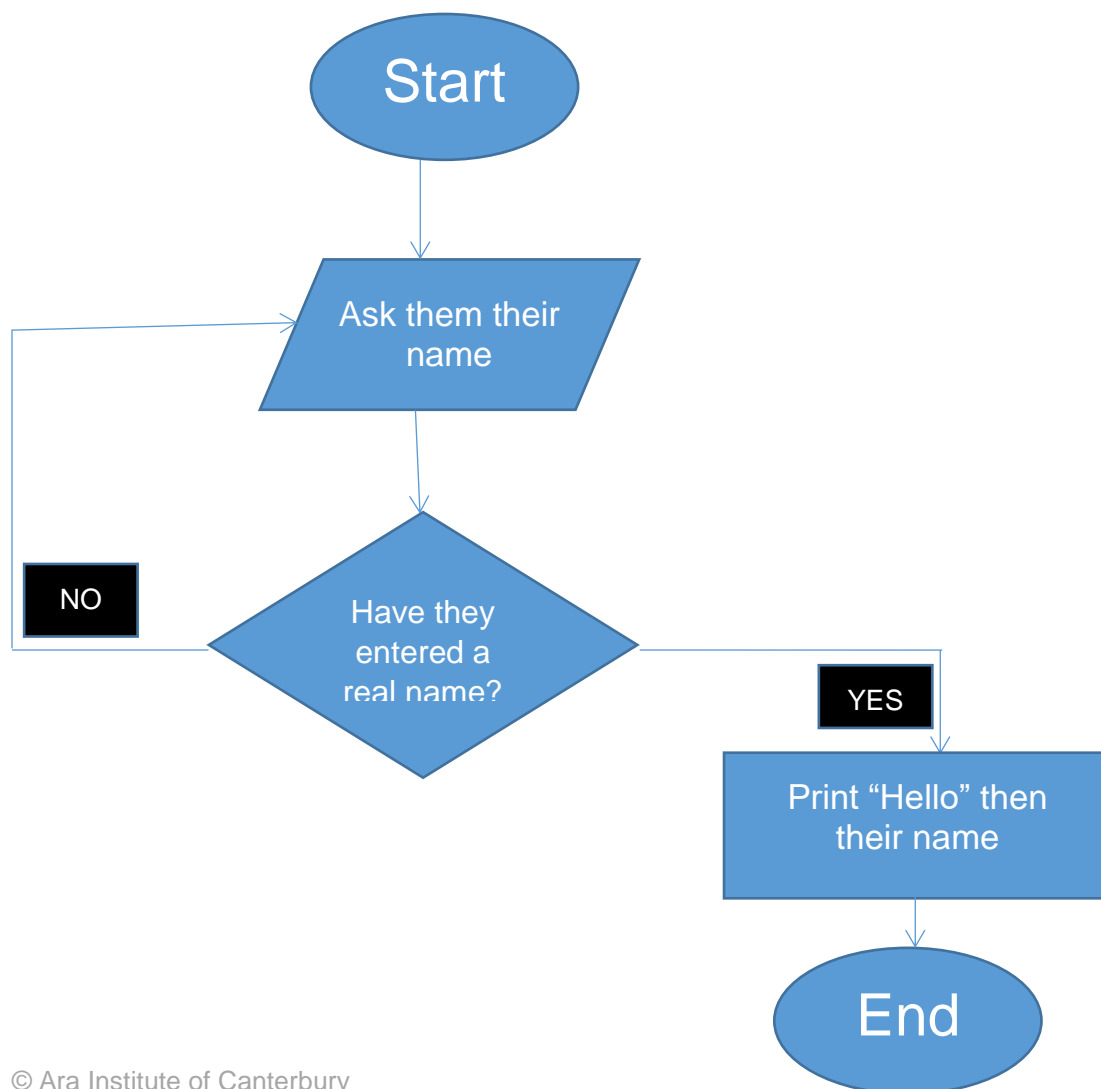
The decision box is also used where there is a loop and a process or processes will be repeated until a certain condition is met:



The example on the previous page shows a flowchart that looks correct but is (practically) impossible to program as it does not define any conditions as to what is or isn't a valid name. If you don't apply a specific condition, how are you going to actually know if a name is valid or not? For this reason, it is important to check that the logic you are outlining in your flowchart is actually feasible.

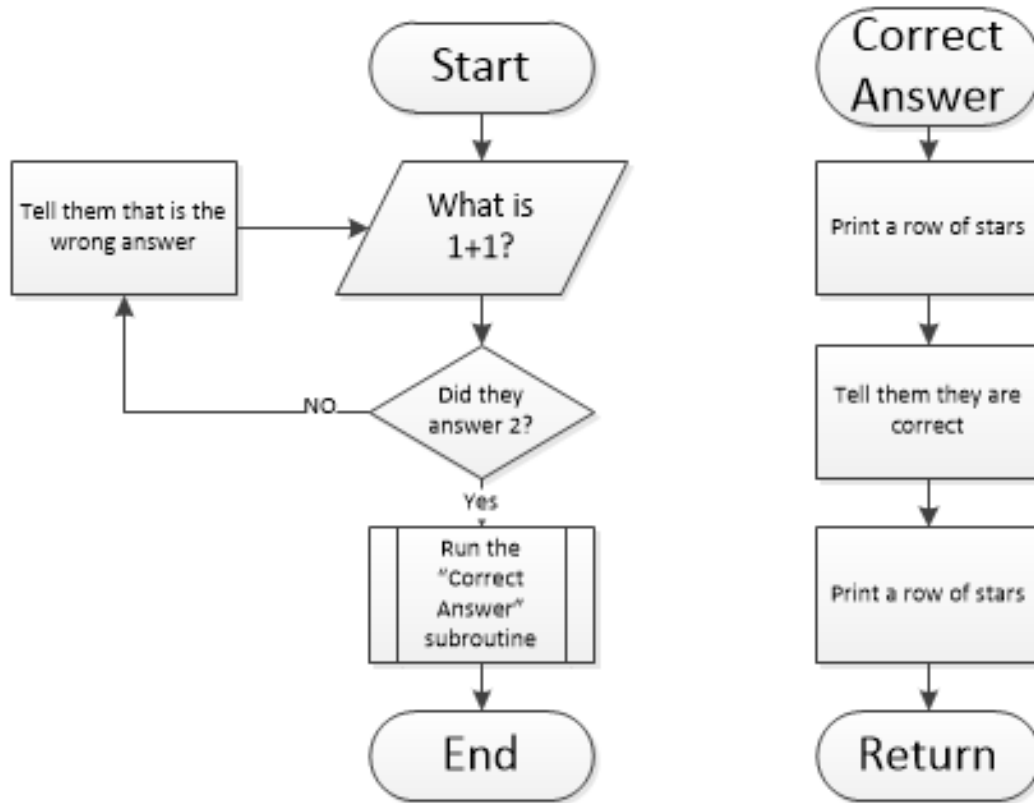
## The end box

- At Level 2, it was true that many of the programs you created had an “end” point at which the program stopped running.
- At Level 3, the program may not end, and instead may simply loop back to the start or to another point. However, if your program does have a specific “end” point at which it finishes running and nothing else happens, then an END box should be used.
- The END box is important because it ensures you have no dead ends in your flowchart, thereby ensuring the program you later create can work if the logical flow is followed.



## Putting it all together

Let's look at an example that uses all of the shapes:



Note that the program above uses a subroutine which is why the diagram is split into separate parts. Speaking of subroutines...

## Subroutines

Subroutines (functions as you may know them) are part programs that are there to perform a particular task. They are used to separate the program into manageable parts (this is known as modular coding).

**Note:** In case you're wondering, Python class methods aren't represented in quite the same way as functions. We'll learn about this later in this worksheet.

Have a look at the simple program below and its output...

```

7% stars.py - D:/Shared from School/AS91...
File Edit Format Run Options Windows Help

def linesofstars():
    counta = 0
    while counta < starcount:
        print("*****")
        counta = counta + 1
    print("")

global starcount
starcount = 1
linesofstars()
starcount = 2
linesofstars()
starcount = 3
linesofstars()
starcount = 4
linesofstars()

Ln: 8 Col: 16

```

```

7% Python Shell
File Edit Shell Debug Options Windows Help

Python 3.2.2 (default, Sep 4 2011, 09:07:29) [MSC v.1500 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
*****
*****
*****
*****
*****
*****
*****
*****
>>>

Ln: 19 Col: 4

```

When there are subroutines, we need to create a separate flowchart for each subroutine, plus one flowchart for the parts of the program that aren't contained in any subroutines. This applies even if all the parts of your code that aren't contained in subroutines do is call the subroutines.

Produce lines of stars
------------------------------

## Tools for creating flowcharts

Other than the subroutines example on page 4, the flowcharts that have been shown so far in this document have been made using the drawing tools in Word. Keep in mind that Word is not really designed for creating complex flowcharts, and you will find it much quicker and easier to use proper flowcharting tools such as Microsoft Visio or [www.draw.io](http://www.draw.io).

If you don't have Visio, the web interface at draw.io is both effective and free!

Check out [this link](#) for a tutorial on the basics of using draw.io. Remember to **always save your work** before closing the draw.io interface, and if you don't have a OneDrive account, you will need to save your work locally to your computer.

## Variable tables

- As mentioned earlier, variables are not included in flowcharts, but are instead shown in variable tables.
- You may not realise it, but variables take quite a bit of explaining...check out the example variable on the next page to see how complex they can be for even a simple program.

Variable Name	Data Type	Scope	Created by	Comments
age	Integer	Local - Main	Input Box	Contains the age of the customer
name	String	Local - Main	Input Box	The first name of the customer
surname	String	Local - Main	Input Box	The last name of the customer
size	Integer	Global	Input Box	The size of the ice cream the customer wants
number	Integer	Global	Input Box	How many ice creams the customer wants
sizelist	List	Global	Fixed	The prices of all sizes of ice creams
sizecost	Integer	Global	Lookup	Cost of the ice cream extracted from the list
totalcost	Integer	Global	Calculated	Cost of ice creams = number times sizecost

**Variable Name:** This is the name you assign the variable - it must be a sensible name

**Data Type:** The type of data such as integer, string, list, etc.

**Scope:** Whether the variable is local or global - if local the subroutine it is used in must be named. If there are no subroutines this column is not needed

**Created By:** Variables can get their content from direct input, calculations, looking up a list etc.

**Comments:** An explanation of what the variable is for in the program

**Important!** Many languages have case sensitive variable names so don't allow Word to capitalise the first letter for you when typing in your variable names

## Testing tables

Whenever a program is written it must be tested. This includes ensuring variables give the expected result. The types of inputs to be tested come under a number of categories:

- Expected Input:** Input values that fall safely within the range of values we might expect. For example, entering 23 and 75 if asked to enter values between 0 or 100.
- Boundary Cases:** Values that are right on the edge of what the program is programmed to accept. For example, entering 0 or 100 if asked to enter a value between 0 or 100.
- Unexpected Inputs:** Values outside the range of what the program is created to accept. For example, entering A, -1, or \$ if asked to enter a value between 0 or 100.

A typical testing table for a flowchart might look like:

Age	Expected Result	Actual Result
0	Program Exits	
3	Told age invalid and returned to input box	
13	Told they are too young	
14	Told they are too young	
15	Told they can go to the dance	
21	Told age invalid and returned to input box	
"Enter"	Told they must enter numbers and returned to the input box	
blah	Told they must enter numbers and returned to the input box	

The "Actual Result" column is left blank during the planning phase, and used when the program has actually been written and is being and tested.

## Flowcharts and Graphical User Interfaces

Now that we have reviewed flowcharts, variable tables, and testing tables, let's take a look a look at how we can apply these processes to programs that make use of graphical user interfaces.

### A simple example

Let's look at a simple program in Python that creates a basic GUI using subroutines:

Program code:



```
7% GUI_1.py - D:\Shared from School\AS91637 - 3.46 Develop a Complex Computer Progra...
File Edit Format Run Options Windows Help

from tkinter import *

def quit():
    root.destroy()

def hello():
    print("Hello")

def main():
    global root
    root = Tk()
    Button(root, text="Quit", activebackground="green", width=8, command=quit).pack()
    Button(root, text="Hello", bg="red", width=8, command=hello).pack()
    root.mainloop()

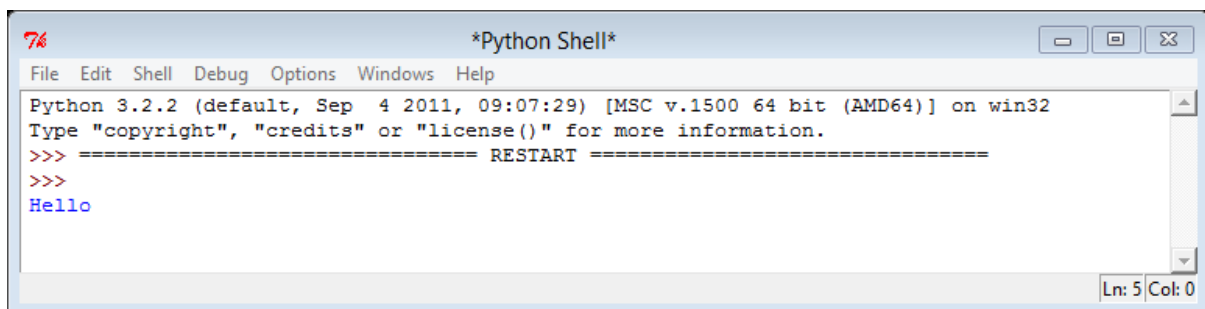
main()

Ln: 9 Col: 11
```

Program running:



Program output:



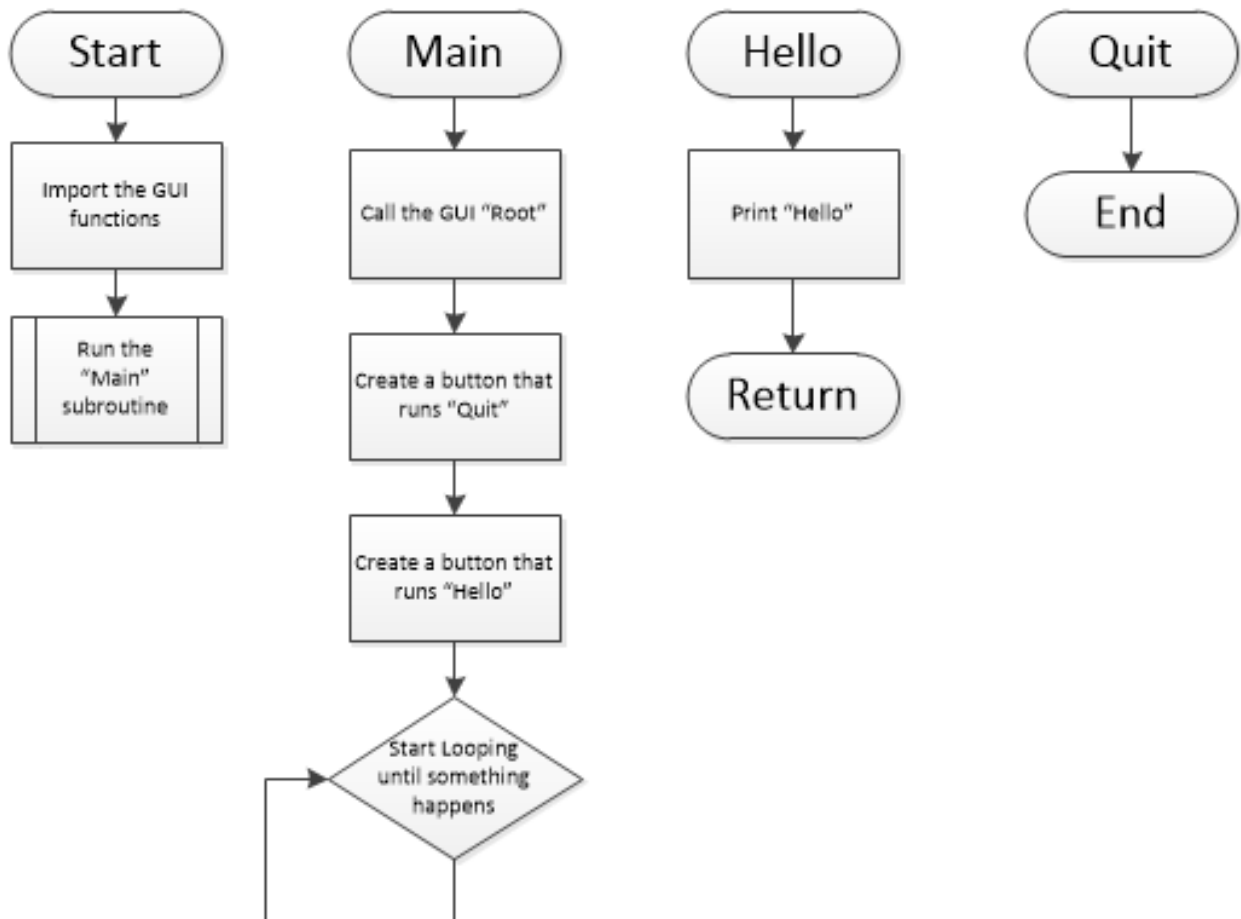
```
7% *Python Shell*
File Edit Shell Debug Options Windows Help

Python 3.2.2 (default, Sep  4 2011, 09:07:29) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello

Ln: 5 Col: 0
```



Below is a sample flowchart for this program:



You may notice a few new things with this flowchart:

- It is a little harder to tell the subroutines from the initial part of the program – the program starts at “Start”.
- When the program starts, all it does is import the GUI and run the “Main” subroutine. This is good practice.
- There is no “End” to the “Start” section of the flowchart - this is because the program never ends here in this initial part.
- The “Main” subroutine finishes with a loop. This is because once “Main” has set up the environment, it just waits until a button is clicked or something else happens (this is the function of the `root.mainloop()` line of code).
- The “Quit” button ends the program, which is why the “End” is grouped with it.

# Task 1 – Create a flowchart for a simple program

Create a flowchart for the program below. Keep in mind that normally we would create the flowchart **before** making the program, however this is just for practice!

*Program code:*

```
from tkinter import *

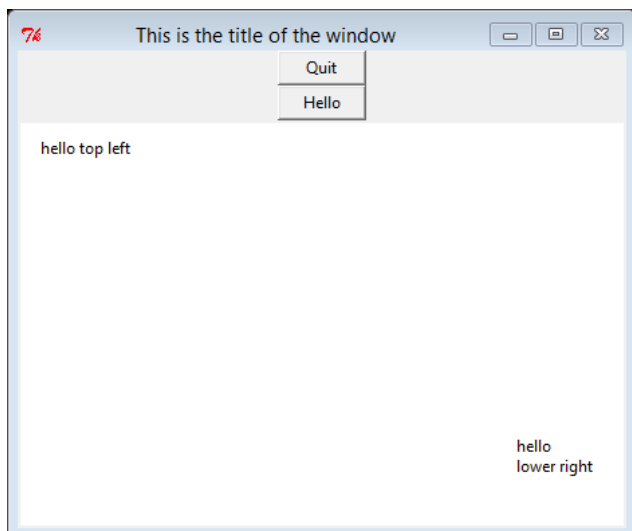
def quit():
    root.destroy()

def hello():
    canvas.create_text(50, 20, text="hello top left")
    canvas.create_text(400, 250, text="hello\nlower right")
    canvas.update()

def main():
    global root
    global canvas
    root = Tk()
    root.title("This is the title of the window")
    Button(root, text="Quit", width=8, command=quit).pack()
    Button(root, text="Hello", width=8, command=hello).pack()
    canvas = Canvas(root, width=450, height=300, bg = 'white')
    canvas.pack()
    root.mainloop()

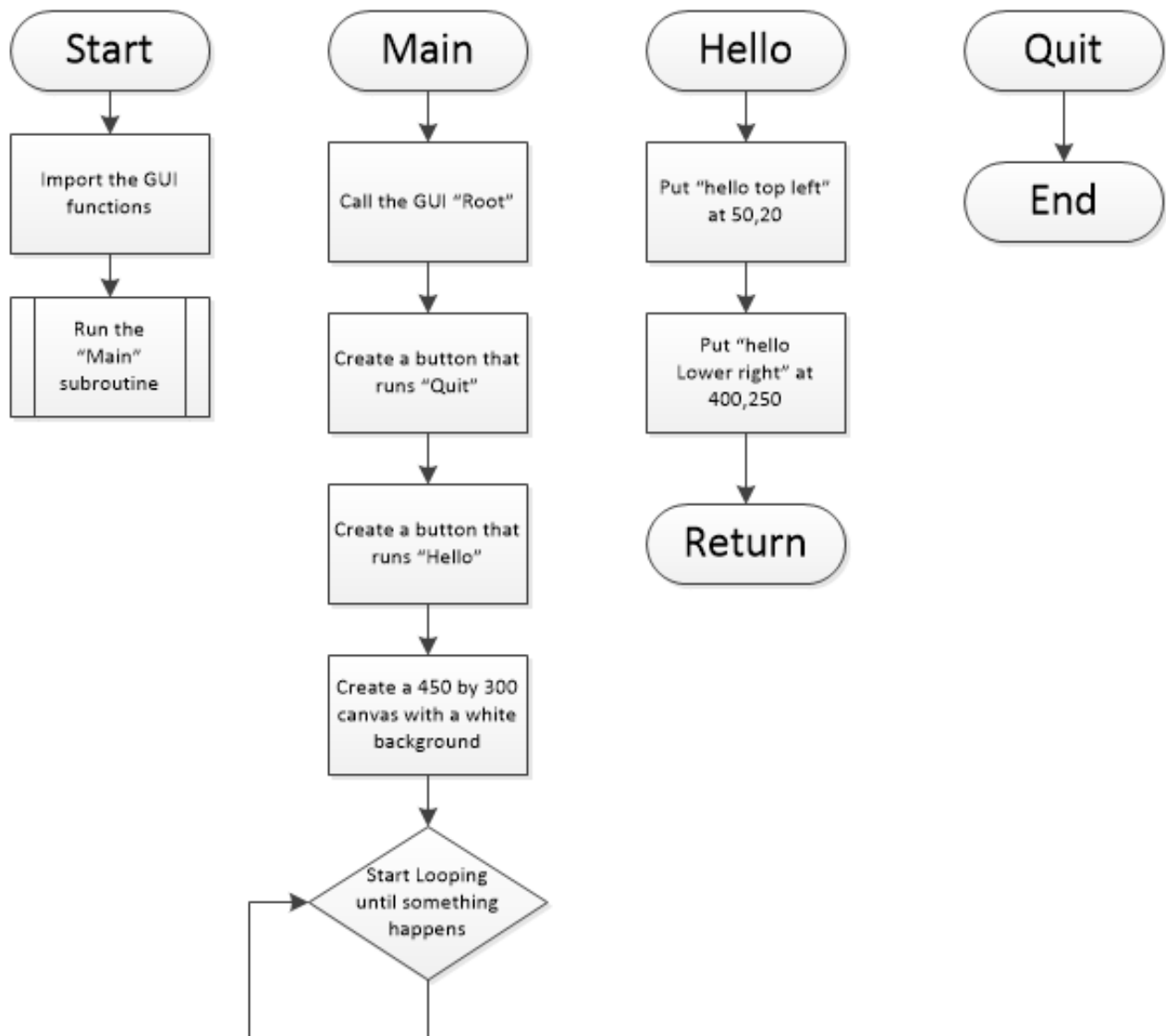
main()
```

*Program running:*



**Don't look** at the answer on the next page until you have tried it yourself...!

Sample answer:



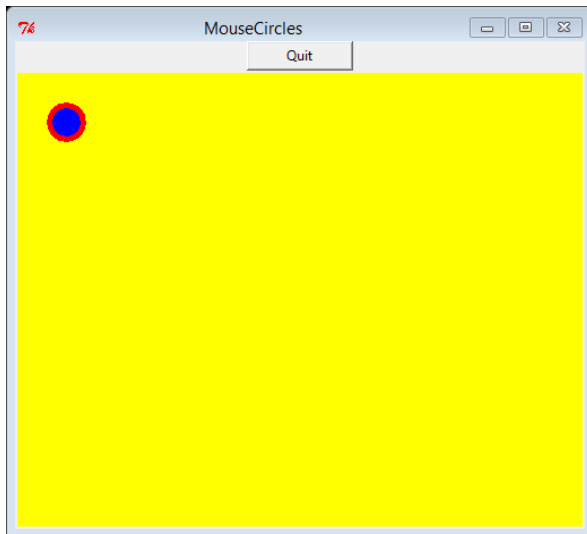
This flowchart is probably more detailed than it needs to be. "Create rectangular canvas" or "put 'hello top left' at the top left" would be fine – but if you know the details, you might as well include them.

Note the inclusion of "**Return**". This is important, as when the program completes the "Hello" subroutine, it logically 'returns' to the Main subroutine until something else happens. Without this Return, the flowchart would imply that the program stopped running after the Hello subroutine completed (which is incorrect).

## Task 2 – Flowchart practice

A program is needed that has a square yellow coloured canvas, and when you click on it, it produces a blue circle with a red border at the point where you click. The window should have the title “Mouse Circles”, and a quit button should be included.

**Produce the flowchart for the program.**



No sample answer is provided for this one, but if you're stuck, look at the sample answer to Task 1 for some guidance. You may also wish to add a process box for window title in your flowchart, which wasn't done in the Task 1 sample answer.

## Classes and objects revisited

Our flowcharts do get a little more complicated when we start to use classes and objects.

Remember a **class** might be something like “Dogs”; another **class** might be “Cats”. An **object** in the dogs' class might be “Spaniel” or “Poodle”.

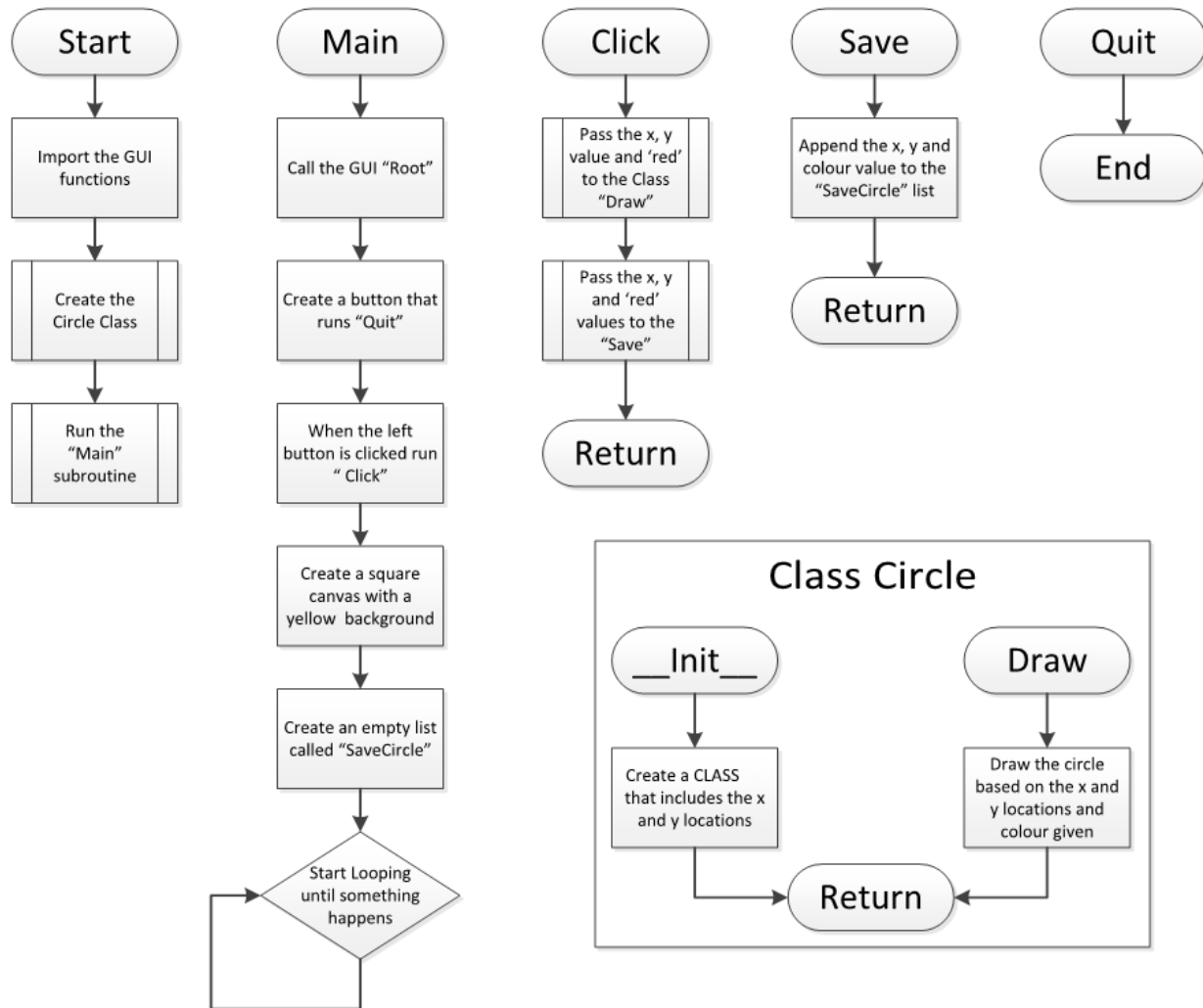
## Lists (or arrays) revisited

Lists allow you to store, retrieve and update related information, for example, names of people.

With classes, lists allow you to store information about various objects in the class so that you can do something to them later.

# Flowcharts with classes, objects, and lists

The flowchart below includes classes, objects, and lists, and there are notes below to explain the features:



- The class is called Circle, and the objects could have different sizes, colours, border widths, etc. Note how the class is defined, and that each object can vary by x, y, and colour.
- Note how the class is put in a container to highlight that it is a class – you can find containers in Visio under Insert/Container. Check out [this link](#) for how to turn any shape into a container using draw.io (it is recommended that you use a rectangle as your base shape).
- See how the empty list is created in the “Main” module.
- Look at the “Click” module – it passes values to other modules.
- This is different to variables that don’t need to be passed, which are not specified in the flowchart but are instead included in the variable table.

Remember that flowcharts just form part of a plan for a program – they do not explain HOW the program is to be written. They do, however, show where modules are located, what class modules do, and describe the flow of the program.

## Task 3 – Adding a subroutine to a flowchart

The flowchart on the previous page is missing the ability to change the colour of all the circles.

Add a subroutine to the flowchart that loops through the list and changes the colour to green, and as it goes, redraws all the circles. The result of this is that all the existing circles will change to green, but any new circles created will be red.

Don't worry about when to stop looping. Just say "Have all records been updated".

The Visio file for the chart above is provided for you on Moodle (**Task 3 – Sample Flowchart.vsd**). If you don't have Visio installed, you can easily double click to view the flowchart in a web browser and then use draw.io to recreate it.

Hint 1: You will need to create an entirely new subroutine, and add a process to the Main function to call this new subroutine.

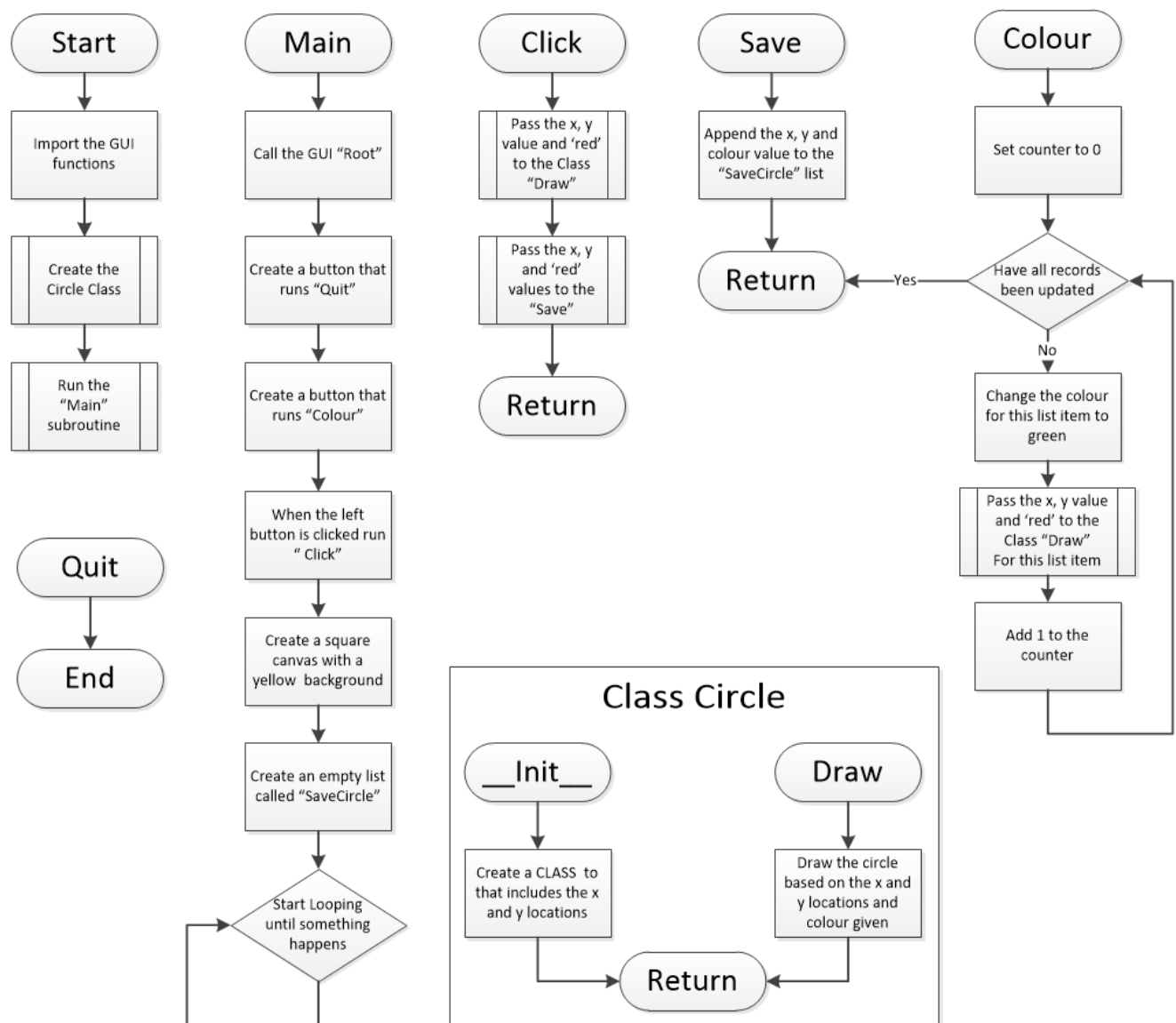
Hint 2: The Start, Click, Save, and Quit sections of your flowchart should remain unchanged.

Hint 3: An easy way to represent looping through something in a flowchart is to have one process box that creates a counter set to 0, and another that adds 1 to the counter.

Hint 3: To make this easier, put your new subroutine to the right of the Save subroutine (by moving your Quit subroutine out of the way), and if all records have been updated, have it link back to the Return at the end of the Save subroutine.

The sample answer is on the next page, but see if you can solve it before taking a look.

Sample answer:



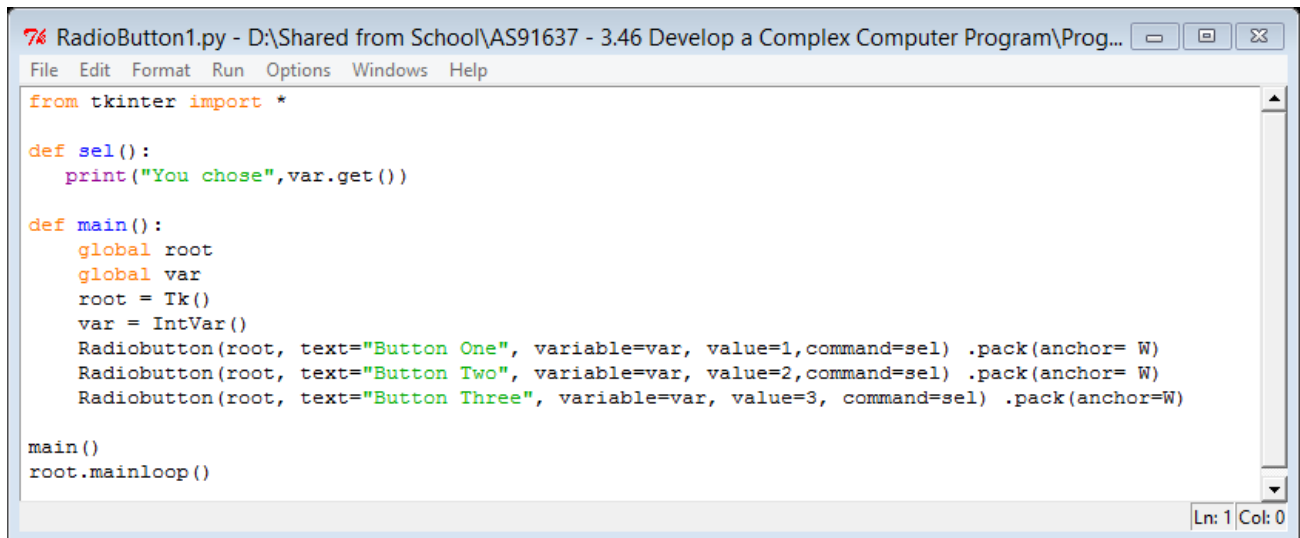
See how the “Draw” method (within the Circle class) is called by the new subroutine to update the circles.

It may have been a good idea to clear the canvas first before the circles were re-drawn, but as they are the same size in the same location it was not required – in other cases it might have been. Keep this in mind when flowcharting your program – if you intend the canvas to be cleared at any point, then ensure that you reflect that logic in your chart as well.

# Radio buttons

We haven't yet learned how to code radio buttons, however as far as flowcharting is concerned, there is typically an action as soon as you click on the radio button. For example:

*Code showing radio buttons:*



```
7% RadioButton1.py - D:\Shared from School\AS91637 - 3.46 Develop a Complex Computer Program\Prog...
File Edit Format Run Options Windows Help

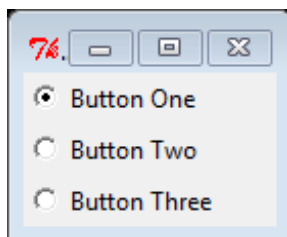
from tkinter import *

def sel():
    print("You chose",var.get())

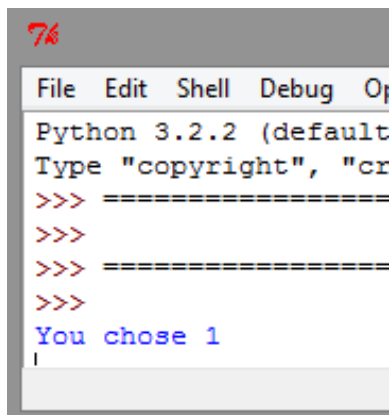
def main():
    global root
    global var
    root = Tk()
    var = IntVar()
    Radiobutton(root, text="Button One", variable=var, value=1,command=sel) .pack(anchor= W)
    Radiobutton(root, text="Button Two", variable=var, value=2,command=sel) .pack(anchor= W)
    Radiobutton(root, text="Button Three", variable=var, value=3, command=sel) .pack(anchor=W)

main()
root.mainloop()
```

*GUI with radio buttons:*

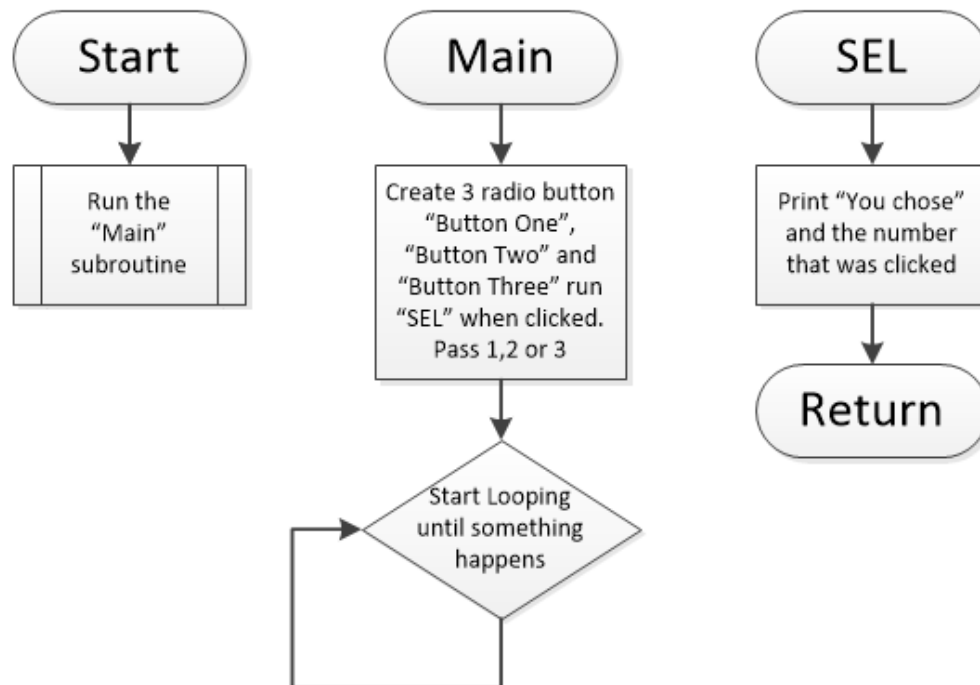


*Potential output from clicking radio button:*





Flowchart for radio button program:



## Check boxes

Again, we haven't yet learned to code checkboxes, however a variable is normally set for each of the buttons you click on and this needs to be reflected in your flowcharts.

Code showing check boxes:

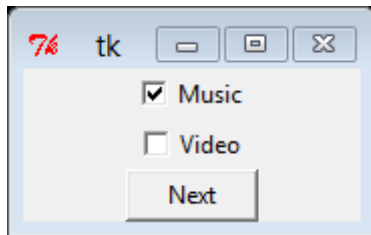
```
7% CheckButton1.py - D:\Shared from School\AS91637 - 3.46 Develop a Complex C...
File Edit Format Run Options Windows Help
from tkinter import *

def main():
    global root, Check1, Check2
    root = Tk()
    Check1 = IntVar()
    Check2 = IntVar()
    C1 = Checkbutton(root, text = "Music", variable=Check1, width = 20) .pack()
    C2 = Checkbutton(root, text = "Video", variable=Check2, width = 20) .pack()
    Button(root, text="Next", width=8, command=nextbutton).pack()
    root.mainloop()

def nextbutton():
    print("For music you choose " + str(Check1.get()))
    print("For video you choose " + str(Check2.get()))

main()
Ln: 1 Col: 0
```

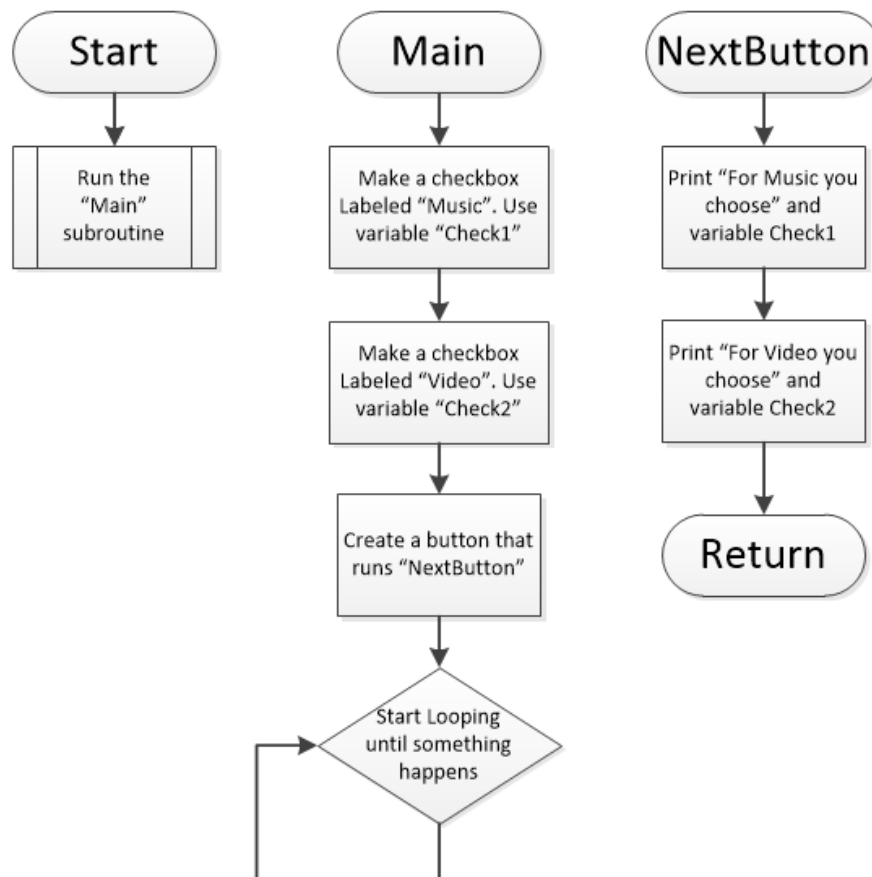
GUI with check boxes:



Potential output from clicking check boxes:

```
Python 3.2.2 (default, Sep 32
Type "copyright", "credits"
>>> =====
>>>
For music you choose 1
For video you choose 0
|
```

Flowchart for check box program:



# Labels, text boxes and images

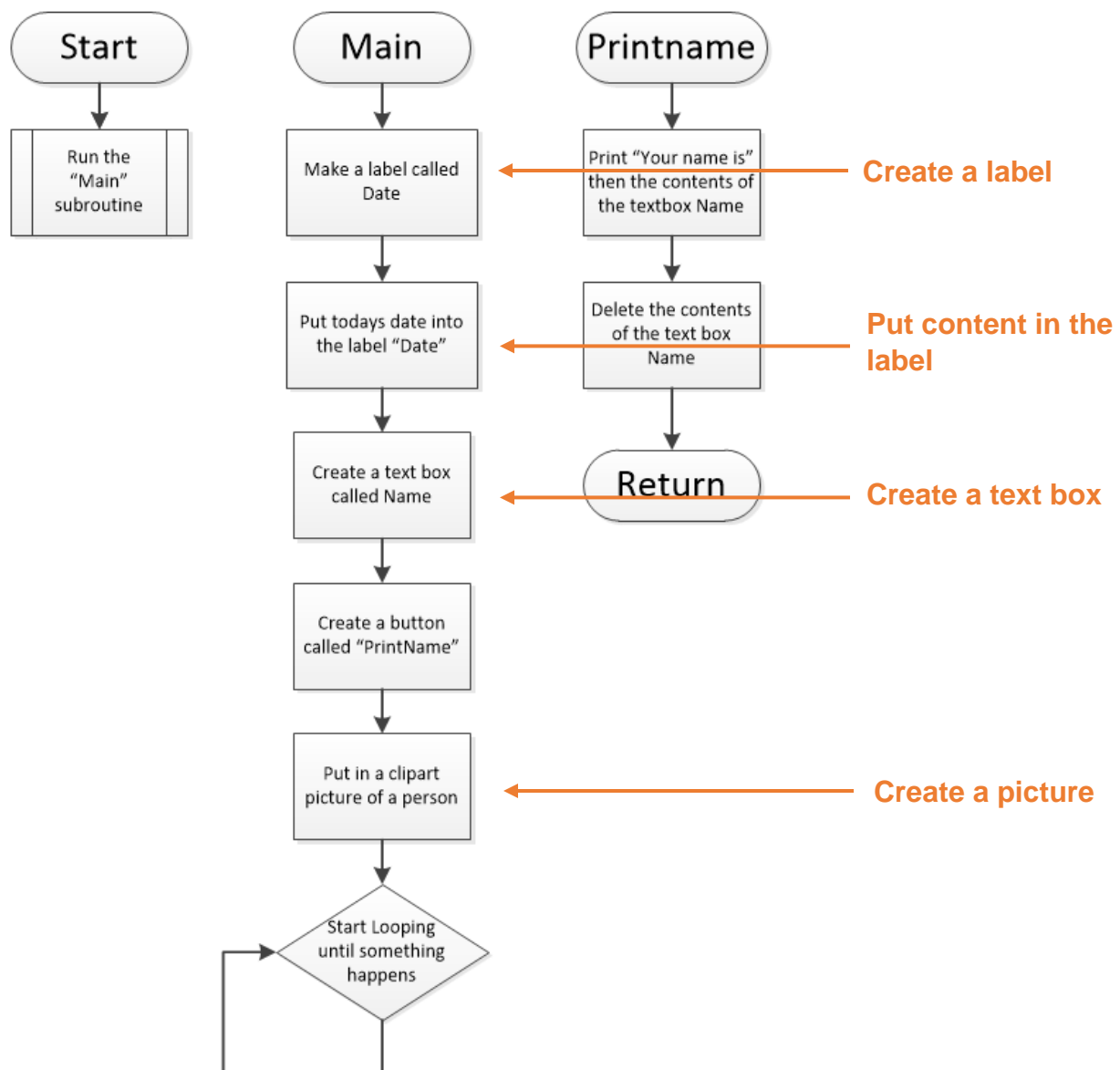
Labels are boxes on a GUI that have text put into them by the program. You **cannot** type into them.

Text boxes on the other hand are boxes on a GUI that you **can** type text into.

The program can often also put text into a text box, but a label should be used if the box will **never** be typed into.

Images also don't actually run any processes – you just need a box in the flowchart to tell you to put in the image.

Because labels, text boxes, and images don't run any processes, we simply represent them in process boxes as shown below:



## More complicated flowcharts

On the next page is the flowchart to produce a basic calculator similar to the one you created in Worksheet 5 (Week 3 of this course).

Each subroutine or module is quite simple – there are not that many boxes in each. This is one of the reasons why working in modules is such a good idea, as it lets you break down the program into the component parts.

Remember one box is not one line of code, but it is one idea or process for which you will write code.

## The perfect flowchart

A flowchart is produced before the program is created. There is, however, no expectation that it will be perfect.

The flowchart will often need to be revised as the program is written.

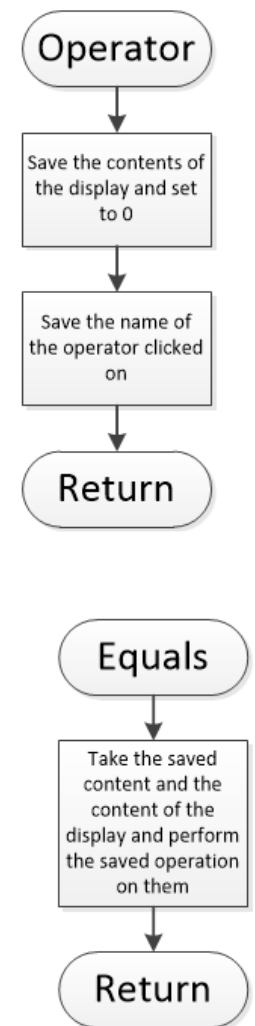
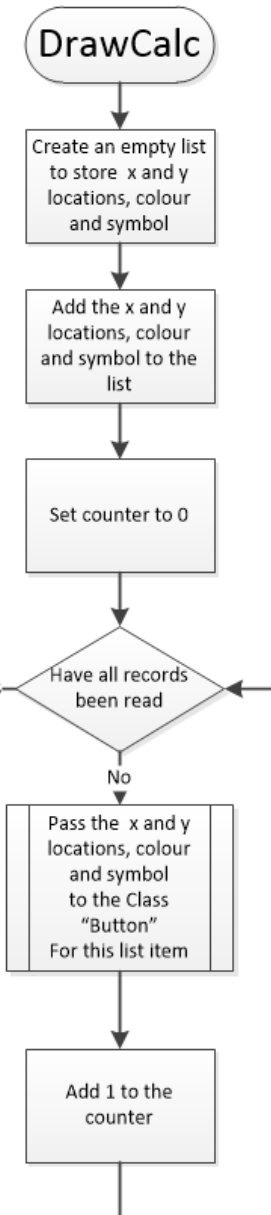
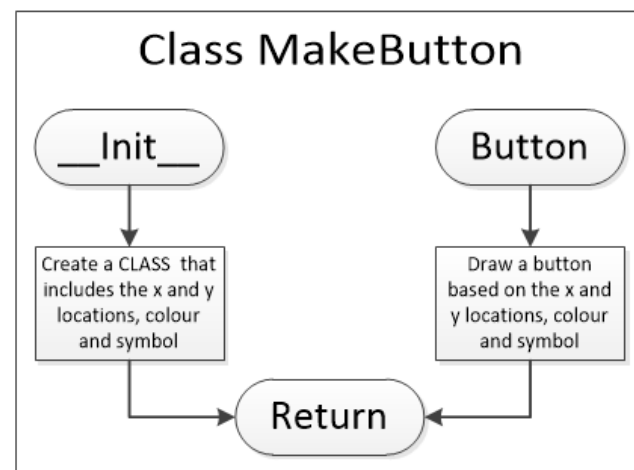
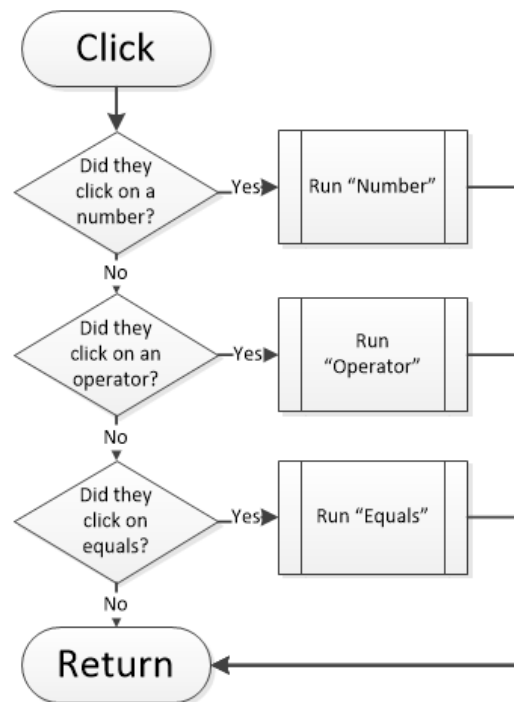
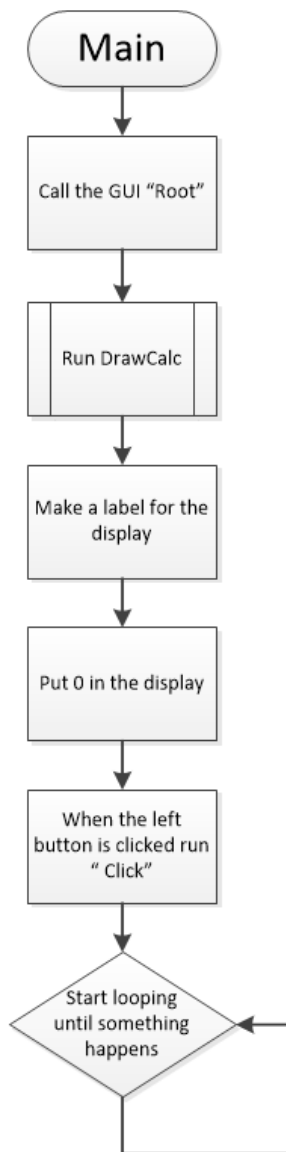
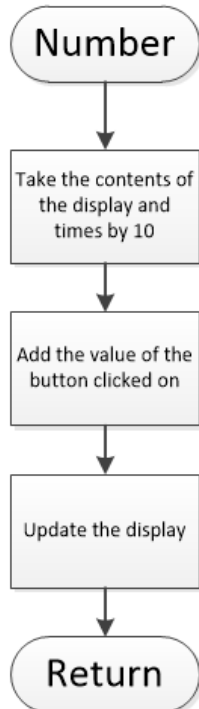
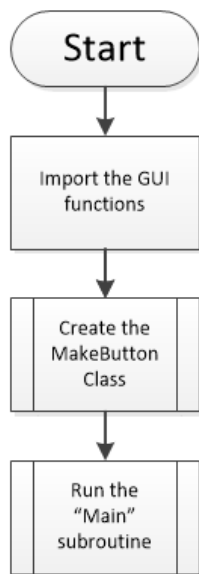
Before you start to program, the flowchart **must appear** to be correct. You and your teacher will probably not be able to see where all changes will be needed before the program is created, however this does not mean that you can hand in a poorly developed flowchart saying that you will fix it up after you write the program.

## Task 4 – find the mistake

The flowchart on the next page is not perfect – have a look at it and try and find the problem.

A program based off this flowchart has not yet been written so *any and all* potential mistakes could be difficult to work out, however there is one very important feature missing.

Once you have found it, have a look on page 22...



## Mistake

Did you find it?

The mistake is that there is no 'Clear' button – the program would need to be restarted after each calculation for it to work properly.

## Variable table for the calculator

Variable Name	Data Type	Scope	Created by	Comments
MakeButton	Class/objects	Global	Fixed	Defines the buttons used in the calculator
Buttons	List	MakeButton	Fixed	Contains the details for each button object
Counter	Integer	Local	Calculated	Allows items in the list to be read in order
Store	Integer	Global	Operator	Holds the previous value of the display

Note how the scope of the Buttons List shows that it holds the values for the MakeButton class. Also note that the data type for MakeButton is class/objects.

## We need to draw

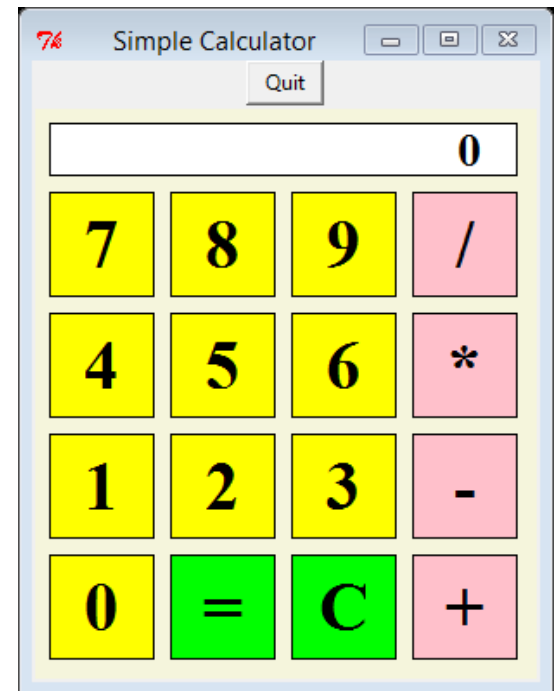
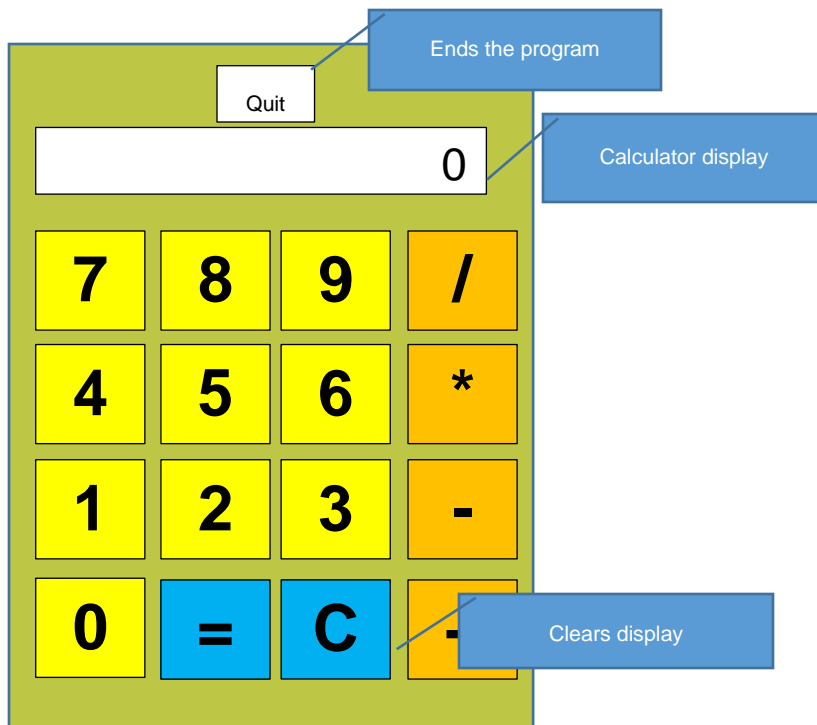
The variable table above correctly shows all the variables in the calculator. It is not, however, enough to create a calculator.

A drawing is also needed, which can be created with pen and paper, or created on the computer.

There is one rule to keep in mind when planning, and that is that you **cannot** create the plan using the programming language – if you do you are not planning, you are programming.

Below to the left is a drawing of a calculator made with Paint. Below to the right is the completed program in Python.

Note that you must also add labels showing what each part of the diagram will do (the labelling below is incomplete).



## Looking at the assessment requirements

In the assessment task, there are many possible solutions. It is, however, important that you give yourself every opportunity to pass with the highest possible grade.

Let's look beyond the obvious requirements like needing objects, classes, and a GUI and see what else is needed:

### Well Structured Modules

- Make sure you separate your program into a number of modules/subroutines.
- Have a module called 'main', and once the program is initialised, call that so the entire program is made of subroutines.
- Ensure each subroutine/module does one task only. If it is doing too much then split it into separate subroutines.
- Each module must have a simple name that describes what it does.

### Event Handling

- Event handling is basically what will happen when you click or enter something.

- The program must be able to handle all events such as clicking repeatedly on the same spot or invalid input.
- It should also do what is expected when the program is used as expected.

## Documentation

- Your diagrams must be fully labelled, showing what they do and what variables are used in them.
- The program must be fully documented to explain what each subroutine does, plus all sections of code.
- The program must also be laid out in a clear and easy to understand way.
- Where necessary, write paragraphs to explain your decisions used to produce the plan and program.

## Testing

- The program needs to be tested for expected boundary and unexpected variables.
- If you create a program that **cannot** have boundary or unexpected variables you may find you only can get Achieved.
- The calculator only has places where you click so you cannot have boundary or unexpected variables. As a result, no matter how good the plan and program, you could only get Achieved.
- To gain a higher grade, the program would need to be modified, i.e. you can type in the numbers as well as clicking on the buttons.