

# technoteach

## technocamps



Llywodraeth Cymru  
Welsh Government



Prifysgol  
Abertawe  
Swansea  
University



Cardiff  
Metropolitan  
University

Prifysgol  
Metropolitan  
Caerdydd



University of  
South Wales  
Prifysgol  
De Cymru

Cyngor Cyllido Addysg  
Uwch Cymru  
Higher Education Funding  
Council for Wales

hefcw



Prifysgol Cymru  
Y Drindod Dewi Sant  
University of Wales  
Trinity Saint David



PRIFYSGOL  
ABERYSTWYTH  
UNIVERSITY

PRIFYSGOL  
Glyndŵr  
Wrexham

PRIFYSGOL  
Wrexham  
glyndŵr  
UNIVERSITY

institute of  
CODING  
in wales technocamps

# Python: What Is it? Who Uses it?



# What Is it? Who Uses it?

Python is an interpreted, high-level, general-purpose **programming** language.



Disney

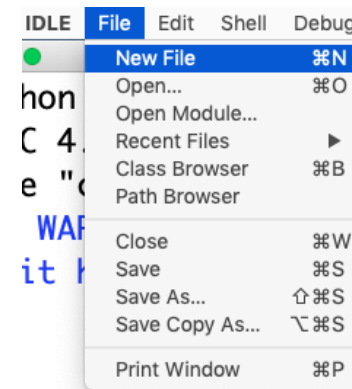
Google

You Tube



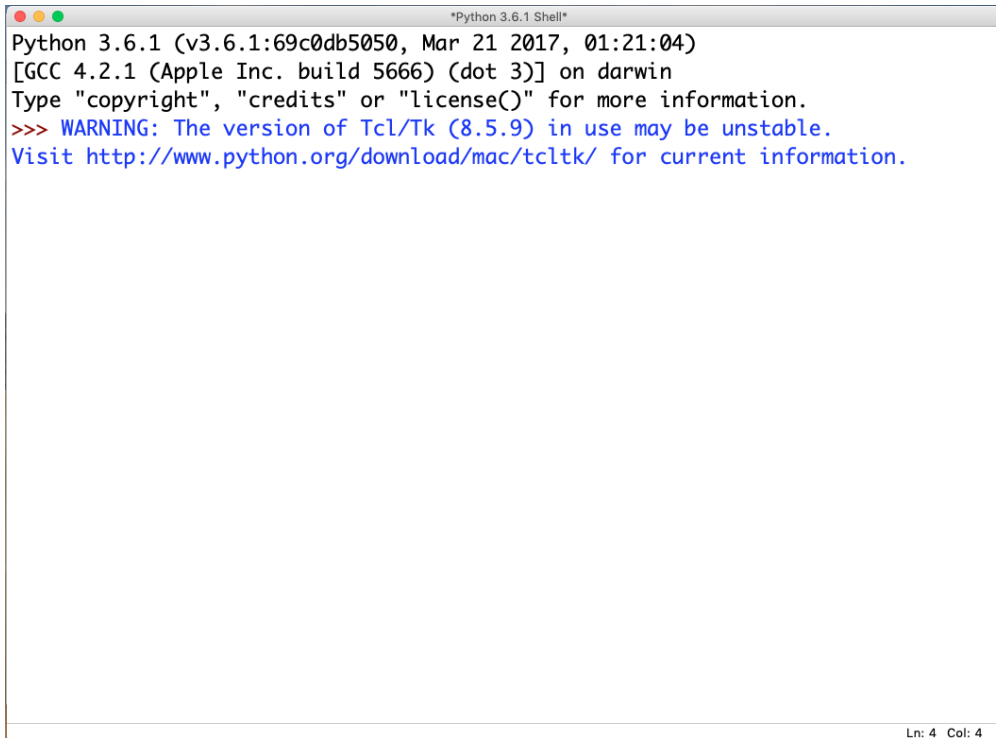
# Activity: Output

1. Open Python Idle (make sure it's Python Version 3)



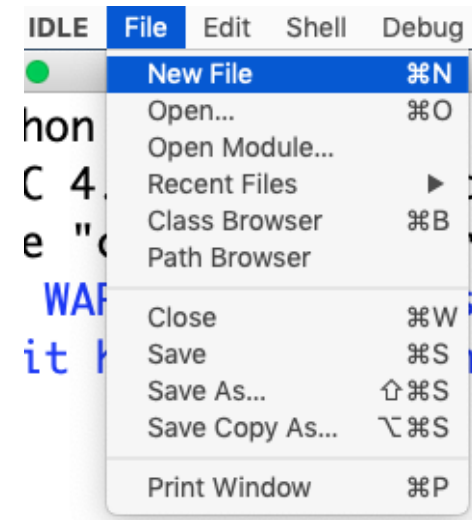
# Activity: Output

2. Once it loads create a new file:



```
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
```

Ln: 4 Col: 4



# Activity: Output

3. Copy the program below.

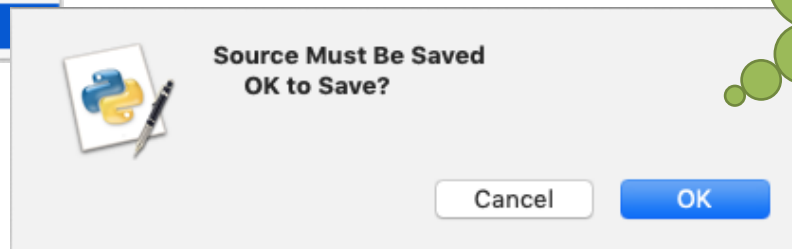
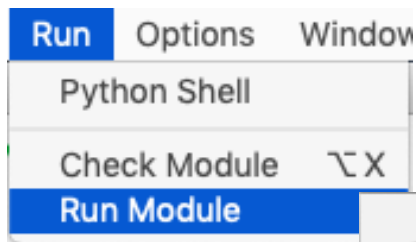
```
print("hello world! ")
```

```
print(2+2)
```

```
print("2+2")
```

# Activity: Output

- Run your program by clicking Run > Run Module  
It may ask you to save your code first.



```
Python 3.6.1 Shell
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

===== RESTART: /Users/caseydenner/Downloads/test.py =====
hello world
4
2+2
>>>
```

# Review Output

What was the outcome of running the program?

What was the difference between line 2 and 3?

```
Python 3.6.1 Shell
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

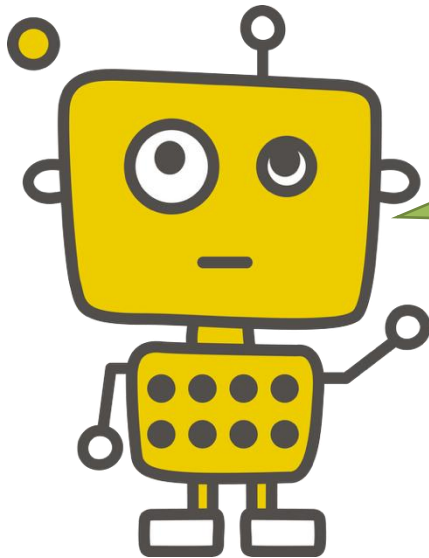
===== RESTART: /Users/caseydenner/Downloads/test.py =====
hello world
4
2+2
>>>
```



# Variables

Variables are a storage placeholder for texts and numbers.

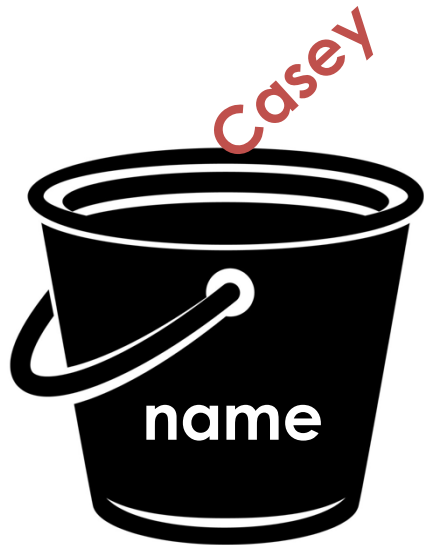
Variables should have logical names as they can be read and edited further on in the code.



Variables should not contain spaces! They should also use camelCase. E.g.  
thisIsCamelCase  
myAge  
myName

# Variables

```
name = "Casey"
```



```
name = "Luke"
```



# Variables

```
age = 20
```

```
name = "Casey"
```

```
print(name, "is", age, "years old")
```

Variables

## Syntax

Variables in python should be in camelCase. E.g. firstName, lastName, etc. They should also be named sensibly!

# Activity: Hello \_\_\_\_\_

Write a Python program that stores your name as a variable and then prints "Hello *yourname*".

# Data Types

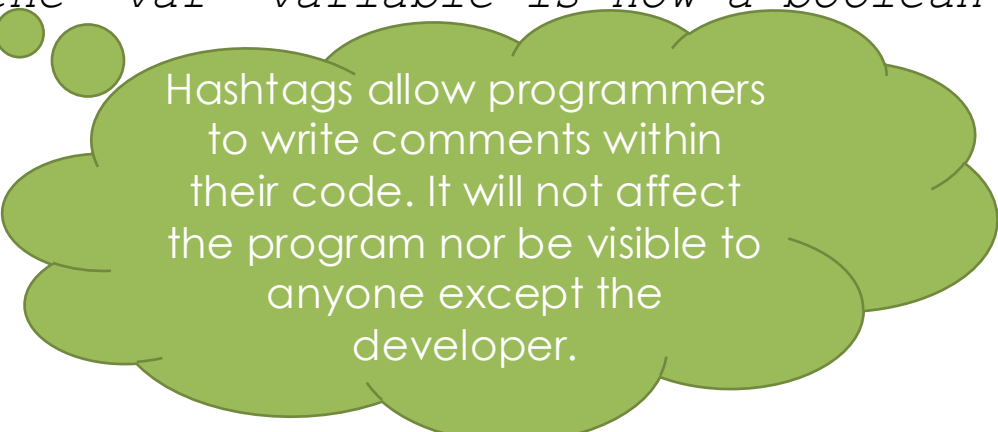
Python sets the variable type based on the value assigned to it. It is a **dynamically typed language**.

For example:

```
var = 10 #This will create a number integer assignment
```

```
var = "text" #the `var` variable is now a string type.
```

```
var = True #the `var` variable is now a boolean type.
```



Hashtags allow programmers to write comments within their code. It will not affect the program nor be visible to anyone except the developer.

# If Statements

If statements are conditional statements.

If something is true it will execute the next line of code.

```
age = 15  
if age>=17:  
    print("you can drive if you have a  
licence")
```

What would happen if the above code was executed?

# If Statements

What would happen if the below code was executed?

```
age = 17
```

```
if age >= 17:
```

```
    print("you can drive if you have a  
license")
```

# If-Else Statements

```
age = 15  
if age>=17:  
    print("you can drive if you have a license")  
else:  
    print("you are not old enough to drive")
```



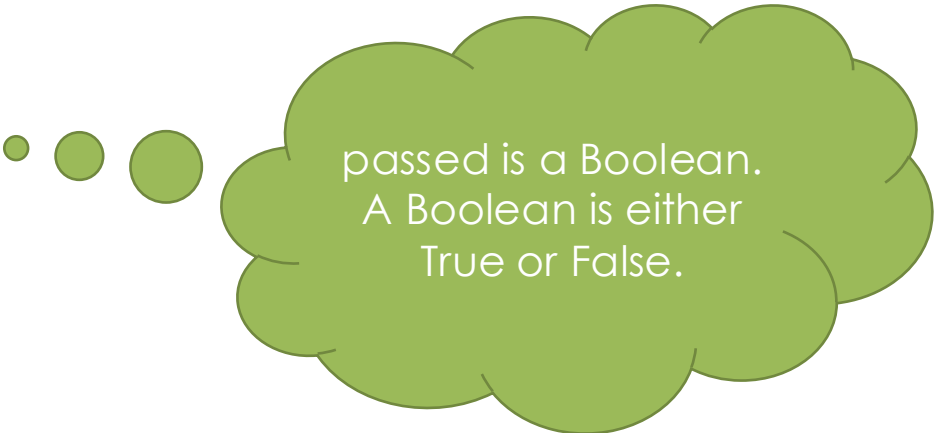
# If-Elif Statements

```
age = 15  
if age>=17:  
    print("you can drive a car if you have a license")  
elif age==16:  
    print("you can drive a moped if you have a license")  
else:  
    print("you're too young to drive anything")
```

# If Statements Using “and” and “or”

If statements can also use “and” and “or” to compare values.

```
age = 17
passed = False
if age>=17 and passed:
    print("You can drive")
elif age>=17 and not(passed):
    print("You can apply to do your test")
else:
    print("You're too young to drive")
```



passed is a Boolean.  
A Boolean is either  
True or False.

# If Statements Using “and” and “or”

If statements can also use “and” and “or” to compare values.

```
a=10
```

```
b=9
```

```
c=15
```

```
if a>b or a>c:
```

```
    print("a condition is met")
```

# Activity: Can You Drive?

Write a program that has two variables:

- Your name
- Your age

Using if-elif-else statements check if you're old enough to drive:

- If you are then it should print: `yourname` is `yourage` years old.  
`Yourname` is old enough to drive.
- If you are not old enough then it should print: `yourname` is `yourage` years old. `Yourname` is not old enough to drive.

**Extension:** Use a Boolean to check if the person has passed their test and utilise this to check if they can drive.

# Input

Python can also ask users for input.

```
name = input("What is your name?")  
print("Hello", name)
```


# Activity: Can You Drive? Input

Go back to your if statement program. Edit it to ask for the person's name and age before it prints if they can drive or not.

# PyShop

Throughout these workshops you will be writing lots of programs contributing to your PyShop.

You're welcome to either create new files each time and copy and alter the old information, or you can edit the same file continually.

When you see this symbol:  The activity is related to the overall PyShop.



# Activity: Login

Create a login screen for users. It must do the following:

- Store a username
- Store a password
- Check if the input username **and** password are correct
- If correct print “Welcome”
- If incorrect print “Login failed”



Don't forget to write comments to explain what your program does when needed!



# Activity: Login Solution

```
username = "casey"
password = "secret"

inUsername = input("Please enter the username: ")
inPassword = input("Please enter the password: ")
if inUsername==username and inPassword==password:
    print("Welcome")
else:
    print("Login failed")
```

# Casting Input

Sometimes programs ensure users' input is of a certain type.

For example, if a user enters an ID number, we'd expect them to type out an integer like 1234 not one thousand and thirty four.

# Casting Input

In Python we can cast input to be of a certain type like this:

```
userInput = input("age:")  
age = int(userInput)  
print(age)
```

However if we were to enter a string then it would throw/raise the following error:

```
age:five
```

```
Traceback (most recent call last):
```

```
  File "/Users/caseydenner/Downloads/test.py", line 2, in <module>  
    age = int(userInput)
```

```
ValueError: invalid literal for int() with base 10: 'five'
```

# Exceptions

A program can **handle** or **raise exceptions**.

- When the program handles an exception it deals with it and then continues to run.
- When an exception is raised the program comes to a halt and displays our exception to screen, offering clues about what went wrong.

# Handling Exceptions

To avoid the `ValueError` we need to handle the exception.

- The `try` and `except` block in Python is used to catch and handle exceptions.
- Python executes code following the `try` statement as a “normal” part of the program.
- The code that follows the `except` statement is the program’s response to any exceptions in the preceding `try` clause.

# Handling Exceptions

```
userInput = input("age:")
```

```
try:
```

```
    age = int(userInput)
```

```
    print(age)
```

```
except ValueError:
```

```
    print("you did not enter an integer")
```

```
print("this line was reached")
```

age:five

you did not enter an integer

this line was reached

# Raising Exceptions

Use **raise** to throw an exception if a condition occurs. The statement can be complemented with a custom exception.

# Raising Exceptions

```
month = 2
day = 30
if month==2 and day>=29:
    raise Exception("invalid date")
print("This line was reached")
```



Notice this line  
was not reached.

Traceback (most recent call last):

File "/Users/caseydenner/Downloads/test.py", line 7, in <module>

raise Exception("invalid date")

Exception: invalid date





# Activity: Menu

Extend your login program so that once a user is logged in a menu appears that fulfils the following:

- Gives the user 3 options – 1. View, 2. Add, 3. Delete
- Users must select an option by entering the appropriate number
- When the user enters their option it provides feedback of which option was selected
- If they enter something which is not an option the program should provide feedback
- Input errors must be handled

# Activity: Menu Solution

```
username = "casey"
password = "secret"
inUsername = input("Please enter the username: ")
inPassword = input("Please enter the password: ")

if inUsername==username and inPassword==password:
    print("Welcome")
    userInput = input("Please select an option:
                        \n1.View \n2.Add \n3.Delete \n")
    try:
        selection = int(userInput)
        if selection==1:
            print("You selected view")
```

# Activity: Menu Solution

```
elif selection==2:
    print("You selected add")
elif selection==3:
    print("You selected delete")
else:
    print("Invalid selection")
except ValueError:
    print("Please enter an integer to perform a
        selection")
else:
    print("Login failed")
```

# Loops

Two types of loops exist:

- For Loops
- While Loops

# For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

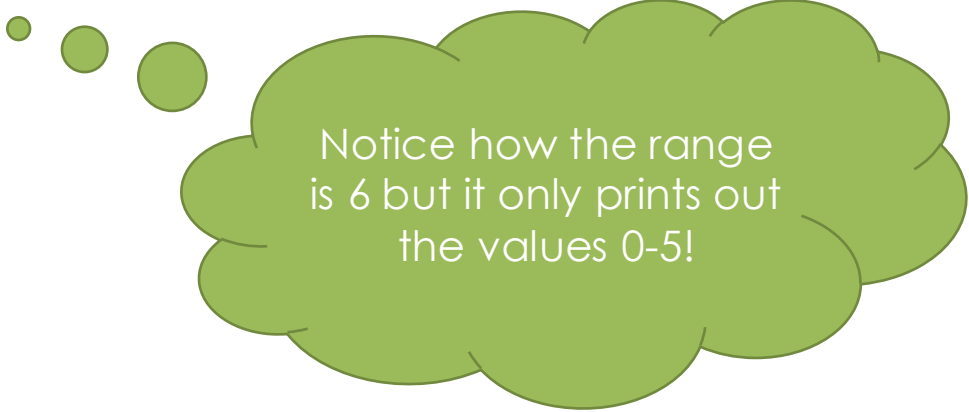
```
fruits =
["banana", "apple", "strawberry"]
for f in fruits:
    print(f)
```

banana  
apple  
strawberry

# For Loops

```
for x in range(6):  
    print(x)
```

0  
1  
2  
3  
4  
5



Notice how the range is 6 but it only prints out the values 0-5!

# While Loop

A while loop can execute a set of statements as long as it is true.

```
password=""  
while password!="secret":  
    password=input("What is the password?\n")  
print("You guessed it!")
```

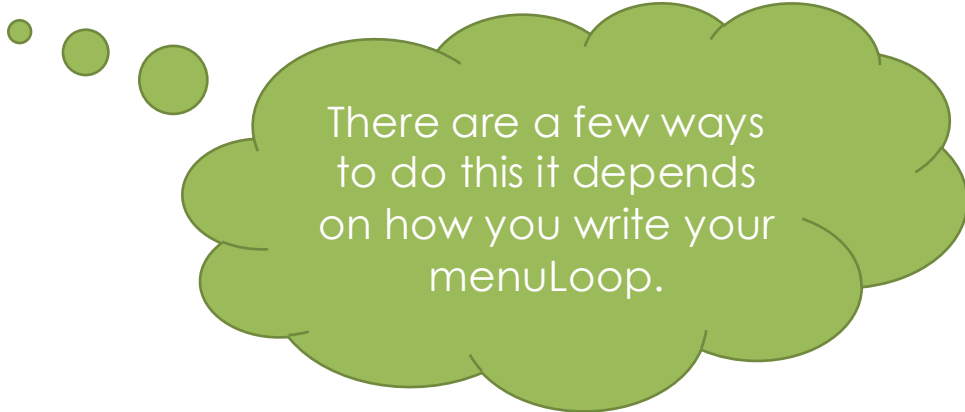
```
What is the password?  
pythonisfun  
What is the password?  
technocamps  
What is the password?  
secret  
You guessed it!
```



# Activity: Login and Menu Loop

Edit your program so that the login and the menu loops if incorrect input is given. For example:

- If the user enters the wrong password and username they should be prompted to input again.
- If the user enters a number above 3, or some text, for the menu selection they should be prompted to enter again.



There are a few ways to do this it depends on how you write your menuLoop.



# Activity: Login and Menu Loop Solution

```
username = "casey"
```

```
password = "secret"
```

```
loggedIn = False
```

```
while not(loggedIn):
```

```
    inUsername = input("Please enter the username: ")
```

```
    inPassword = input("Please enter the password: ")
```

```
    if inUsername==username and inPassword==password:
```

```
        print("Welcome")
```

```
        loggedIn = True
```

# Activity: Login and Menu Loop Solution

```
menuLoop = True
while (menuLoop) :
    userInput = input("Please select an option:
\n1.View \n2.Add \n3.Delete \n")
    try:
        menuLoop = False
        selection = int(userInput)
        if selection==1:
            print("You selected view")
        elif selection==2:
            print("You selected add")
```

# Activity: Login and Menu Loop Solution

```
elif selection==3:
    print("You selected delete")
else:
    print("Invalid selection")
    menuLoop = True
except ValueError:
    print("Please enter an integer to perform a
selection")
    menuLoop = True
else:
    print("Login failed")
```

# Functions

There are often sections of the program that we want to re-use or repeat. Chunks of instructions can be given a name - they are called **functions**.

- Programming languages have a set of **pre-defined** (also known as built-in) functions and procedures.
- If the programmer makes their own ones, they are **custom-made** or **user-defined**.

# Functions

Parameters  
can be added  
and used in  
functions

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

```
print_hello()
```

Hello

```
def print_welcome(name):  
    print("Welcome", name)
```

```
print_welcome("Casey")
```

Welcome Casey

## Syntax

Functions in python should be lower case and each word should be separated by an underscore. E.g.  
do\_something(),  
print\_hello(), etc.

# Functions Execution

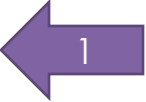
```
def print_welcome(name) :  
    print("Welcome", name)
```

```
print_welcome("Casey")  
print("Done")
```

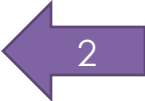



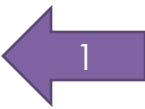
# Functions Execution

```
def print_welcome(name) :   
    print("Welcome", name)
```

```
print_welcome("Casey")   
print("Done")
```

# Functions Execution

```
def print_welcome(name) : 
    print("Welcome", name)  Welcome Casey

print_welcome("Casey") 
print("Done")
```




# Functions Execution

```
def
```

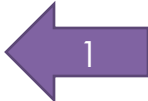
```
print_welcome(name) :
```



```
print("Welcome", name)
```


Welcome Casey

```
print_welcome("Casey"  
)
```



```
print("Done")
```


Done

# Activity: Name and Age Function

- Create a program that asks the user for their name and age.
- Once the name and age has been input you need to pass it to a function that then prints out:
  - “[name] is [age] years old”.



Discuss: How  
can we alter  
our code to  
use functions?



# Activity: Functions

Edit your existing login/menu program to include functions.

Consider the following:

- Do you still need the while loops?
- Can you use function calls instead?

Hints:

- Login can be a standalone function that calls the menu function
- The menu can be a standalone function also

# Activity: Functions Solution

```
username = "casey"  
password = "secret"
```

```
def login() :
```

```
    inUsername = input("Please enter the username: ")
```

```
    inPassword = input("Please enter the password: ")
```

```
    if inUsername==username and inPassword==password:
```

```
        print("Welcome")
```

```
        menu()
```

```
    else:
```

```
        login()
```

# Activity: Functions Solution

```
def menu():  
    userInput = input("Please select an option: \n1.View \n2.Add  
                    \n3.Delete \n")  
  
    try:  
        selection = int(userInput)  
        ...  
    else:  
        print("Invalid selection")  
        menu()  
  
except ValueError:  
    print("Please enter an integer to perform a selection")  
    menu()
```

# Activity: Functions Solution

```
login()
```

Don't forget to start the program you'd have to call the login() function!

# Data Structures

- The key role of a computer program is to store and process data.
- Any computer software has a **data model** that defines **what** data will be collected and worked on.
- The **data structure** defines **how** the flow of data is controlled in relation to inputs, processes and outputs.
- Data structures can have two main characteristics. Firstly they can be **static** or **dynamic**, and secondly they can be **mutable** or **immutable**.



# Data Structures: Lists

Lists have many methods associated with them which we can utilise to manipulate/use the data stored within.

# Lists

```
textList = ["a", "b"]
```

```
numberList = [1, 2, 3]
```

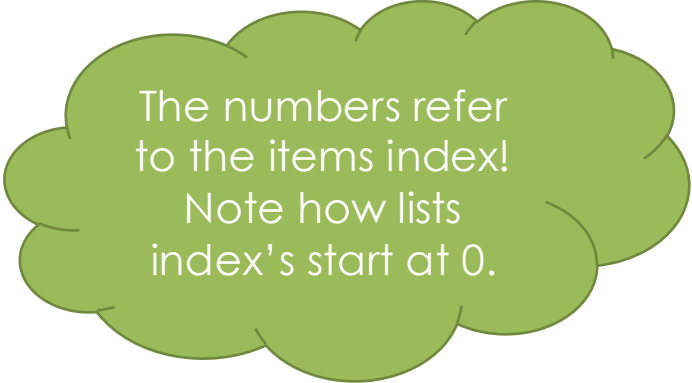
```
mixedList = ["a", 1, "b", 2]
```

# Lists - Accessing an Item

```
textList = ["a", "b", "c"]  
print(textList[1])
```

b

```
textList = ["a", "b", "c"]  
a print(textList[0])
```



The numbers refer  
to the items index!  
Note how lists  
index's start at 0.

# Lists - Getting The Length

```
textList = ["a", "b", "c"]  
print(len(textList))
```

3

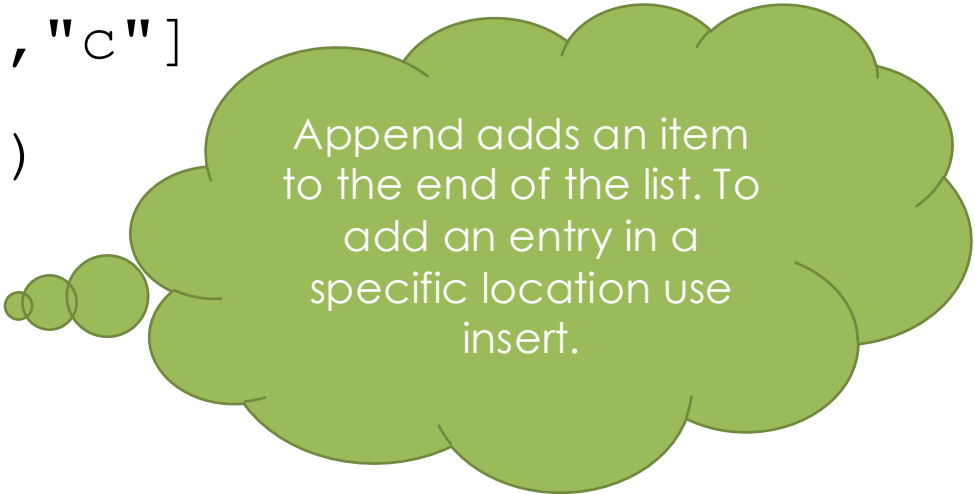
# Lists - Appending

```
textList = ["a", "b", "c"]
```

```
textList.append("d")
```

```
print(textList)
```

```
['a', 'b', 'c', 'd']
```



Append adds an item to the end of the list. To add an entry in a specific location use insert.

# Lists – Editing an Item

```
textList = ["a", "b", "c"]
```

```
textList[0]="z"
```

```
print(textList)
```

```
['z', 'b', 'c']
```

# Lists – Removing an Item

```
textList = ["a", "b", "c"]
```

```
textList.remove("b")
```

```
print(textList)
```


```
['a', 'c']
```

```
textList = ["a", "b", "c", "b"]
```

```
textList.remove("b")
```

```
print(textList)
```

```
['a', 'c', 'b']
```



If the item you want to remove isn't in the list an error is thrown.

If two of the same values are in the list the first one is removed.

# Activity: Lists

- Create a list that holds the names of 5 people.
- Print the list.
- Remove the first item in the list.
- Append another name to the list.
- Print the third element in the list.
- Change the third element to be "Tilly".
- Print the list.
- Print the length of the list.



# Data Structures: Two Dimensional Lists

A list keeps track of multiple pieces of information in linear order, or a single dimension. However, the data associated with certain systems (a digital image, a board game, etc.) lives in two dimensions. To visualize this data, we need a multi-dimensional data structure, that is, a multi-dimensional list.

# Data Structures: Two Dimensional Lists

A two-dimensional list is really nothing more than a list of lists (a three-dimensional list is a list of lists of lists). Think of your dinner. You could have a one-dimensional list of everything you eat:

(lettuce, tomatoes, salad dressing, steak, mashed potatoes, string beans, cake, ice cream, coffee)

Or you could have a two-dimensional list of three courses, each containing three things you eat:

(lettuce, tomatoes, salad dressing) and (steak, mashed potatoes, string beans) and (cake, ice cream, coffee)

# Two Dimensional (2D) Lists

```
menu=[  
    ["prawn cocktail", "spring rolls", "duck pancakes"],  
    ["steak and chips", "chow mein", "chicken tikka masala"],  
    ["ice cream", "chocolate brownie", "strawberry cheesecake"]  
]
```

```
myList = [ [0,1,2,3,4], [3,6,1], [2,1] ]
```

# 2D Lists - Accessing an Item

```
print(menu[1])
```

```
['steak and chips', 'chow mein', 'chicken tikka masala']
```

```
print(menu[0][1])
```

```
spring rolls
```

# 2D Lists - Getting the Length

```
myList = [ [0,1,2,3,4], [3,6,1], [2,1] ]  
print(len(myList))
```

3

```
print(len(myList[0]))
```

5

## 2D Lists - Appending

```
myList = [ [0,1,2,3,4], [3,6,1], [2,1] ]  
print(myList)
```

```
[[0, 1, 2, 3, 4], [3, 6, 1], [2, 1]]
```

```
myList.append([1,6,7,8])  
print(myList)
```

```
[[0, 1, 2, 3, 4], [3, 6, 1], [2, 1], [1, 6, 7, 8]]
```

```
myList[1].append(3)
```

```
[[0, 1, 2, 3, 4], [3, 6, 1, 3], [2, 1], [1, 6, 7, 8]]
```

# 2D Lists - Editing an Item

```
myList = [ [0,1,2,3,4], [3,6,1], [2,1] ]  
print(myList)
```

```
[[0, 1, 2, 3, 4], [3, 6, 1], [2, 1]]
```

```
myList[0]=[1,2,3,4,5]  
print(myList)
```

```
[[1, 2, 3, 4, 5], [3, 6, 1], [2, 1]]
```

```
myList[1][0]=1  
print(myList)
```

```
[[1, 2, 3, 4, 5], [1, 6, 1], [2, 1]]
```

# 2D Lists - Removing an Item

```
myList = [ [0,1,2,3,4], [3,6,1], [2,1] ]
```

```
myList[0].remove(1)
```

```
print(myList)
```

```
[[0, 2, 3, 4], [3, 6, 1], [2, 1]]
```

```
myList.remove([2,1])
```

```
print(myList)
```

```
[[0, 2, 3, 4], [3, 6, 1]]
```



# Activity: 2D Lists

- Create a 2D list that holds 3 starters, 4 main meals and 5 desserts.
- Print the list.
- Remove the first item in the starters list.
- Append another dessert to the dessert list.
- Print the third element in the main meal list.
- Print all of the desserts.
- Print the length of the whole menu.
- Print the length of the starters.

# Data Structures: Dictionaries

Python provides another composite data type called a dictionary, which is similar to a list in that it is a collection of objects.

# Data Structures: Dictionaries

Dictionaries and lists share the following characteristics:

- Both are mutable.
- Both are dynamic. They can grow and shrink as needed.
- Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

# Data Structures: Dictionaries

Dictionaries differ from lists primarily in how elements are accessed:

- List elements are accessed by their position in the list, via indexing.
- Dictionary elements are accessed via keys.

# Dictionaries

```
cars = {  
    "Audi Q5":20000,  
    "Volkswagen Polo":5000  
}
```

```
print(list(cars.keys()))
```

```
['Audi Q5', 'Volkswagen Polo']
```

```
print(cars)
```


```
{'Audi Q5': 20000, 'Volkswagen Polo': 5000}
```

# Dictionaries - Accessing an Item

```
cars = {  
    "Audi Q5":20000,  
    "Volkswagen Polo":5000  
}
```

```
print(cars["Audi Q5"])
```

20000



Dictionary items are accessed by using their keys as opposed to indexes!

# Dictionaries - Getting the Length

```
cars = {  
    "Audi Q5":20000,  
    "Volkswagen Polo":5000  
}
```

```
print(len(cars))
```

2

# Dictionaries - Appending

```
cars = {  
    "Audi Q5":20000,  
    "Volkswagen Polo":5000  
}
```

```
cars["Ford KA"]=500
```

```
print(list(cars.keys()))
```

```
['Audi Q5', 'Volkswagen Polo', 'Ford KA']
```



# Dictionaries - Editing an Item

```
cars = {  
    "Audi Q5":20000,  
    "Volkswagen Polo":5000  
}
```

```
cars["Audi Q5"]=10000  
print(cars["Audi Q5"])  
10000
```

# Dictionaries - Removing an Item

```
cars = {  
    "Audi Q5":20000,  
    "Volkswagen Polo":5000  
}
```

```
cars.pop("Audi Q5")  
print(list(cars.keys()))
```

```
['Volkswagen Polo']
```

# More Advanced Dictionaries

```
cars = {  
    "Q5":{  
        "Brand":"Audi",  
        "Price":20000  
    },  
    "Polo":{  
        "Brand":"Volkswagen",  
        "Price":5000  
    }  
}  
  
print(list(cars.keys()))
```

['Q5', 'Polo']



## Activity: Create a Dictionary/2D List

Edit your login/menu program and create a dictionary/2D list to hold the following information:

Key	Name	Price	Number In Stock
1	Dairy Milk Plain Chocolate	80	15
2	Lucozade Sport	180	10
3	Strawberry Laces	30	8

When a user selects view they should be able to see the contents of the dictionary/2D list.

# Activity: Create a Dictionary Solution

```
username = "casey"
password = "secret"
shopItems = {
    1:{
        "Name":"Dairy Milk Plain Chocolate",
        "Price":80,
        "NumberInStock":15
    },
    2:{
        "Name":"Lucozade Sport",
        "Price":180,
        "NumberInStock":10
    },
```

# Activity: Create a Dictionary Solution

```
3: {  
    "Name": "Strawberry Laces",  
    "Price": 30,  
    "NumberInStock": 8  
}
```

```
def login():
```

```
...
```

# Activity: Create a Dictionary Solution

```
def menu():  
    ...  
    if selection==1:  
        print("You selected view")  
        print(shopItems)  
    ...  
except ValueError:  
    print("Please enter an integer to perform a selection")  
    menu()  
  
login()
```

# Activity: Create a 2D List Solution

```
username = "casey"
password = "secret"
shopItems = [
    [1,"Dairy Milk Plain Chocolate",80,15],
    [2,"Lucozade Sport",180,10],
    [3,"Strawberry Laces",30,8],
]

def login():
    ...
```



# Activity: Create a 2D List Solution

```
def menu():  
    ...  
    if selection==1:  
        print("You selected view")  
        for i in range(len/shopItems):  
            print(shopItems[i])  
    ...  
  
login()
```

# Purchasing From The Pyshop

Shops obviously sell items so people need to be able to complete purchases. When a person purchases an item the total sales value goes up and the amount of stock left goes down. Shop keepers should also be able to view the amount of total sales.

To do this in Python is difficult and you will need to use a global variable.

# Global Variables

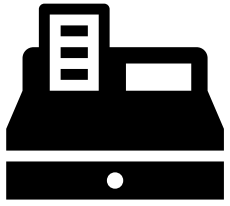
Global variables are the ones that are defined and declared outside a function and we need to use them inside a function.

You only need to use global keyword in a function if you want to do assignments or change the value of the variable. global is not needed for printing and accessing.



# Activity: Purchasing From the Pyshop

The next few slides will contain code that allows purchases to be made. Please copy what you want. You can either copy the dictionary code or the 2D list code.



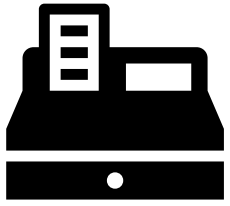
# Activity: Purchasing From the Pyshop

```
username = "casey"  
password = "secret"  
totalSales = 0  
shopItems = {  
    ...
```



Create the  
totalSales variable

A green thought bubble with a black outline, containing the text "Create the totalSales variable". It is connected to the code line "totalSales = 0" by three small green circles.



# Activity: Purchasing From the Pyshop (2D List)

```
def purchase():  
    item = input("Please enter an item key value to purchase")  
    try:  
        key = int(item)-1  
        shopItems[key][3]=shopItems[key][3]-1  
        global totalSales  
        totalSales += shopItems[key][2]  
        menu()  
    except ValueError:  
        print("Please enter a valid key")  
        purchase()
```



# Activity: Purchasing From the Pyshop (Dictionary)

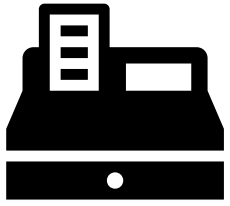
```
def purchase():  
    item = input("Please enter an item key value to purchase")  
    try:  
        key = int(item)  
        shopItems[key]["NumberInStock"] = shopItems[key]["NumberInStock"] - 1  
        global totalSales  
        totalSales += shopItems[key]["Price"]  
        menu()  
    except ValueError:  
        print("Please enter a valid key")  
        purchase()
```



# Activity: Purchasing From the Pyshop

```
def view_sales() :  
    print(totalSales)  
    menu()
```





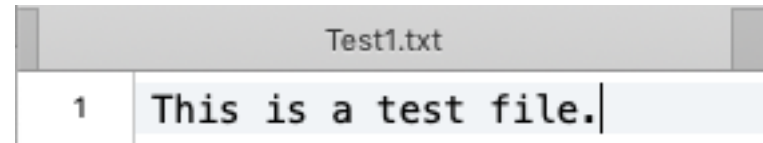
# Activity: Purchasing From the Pyshop

```
def menu():  
    ...  
    elif selection==2:  
        print("You selected purchase")  
        purchase()  
    elif selection==3:  
        print("You selected view sales total")  
        view_sales()  
    ...
```

# Reading From a File

```
f = open("Test1.txt", "r")  
print(f.read())  
f.close()
```

This is a test file.

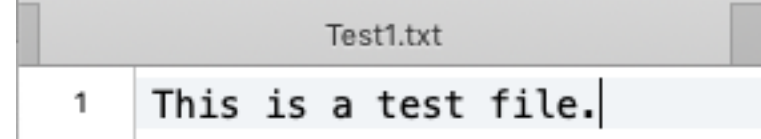


r means read.

# Reading Parts of a File

```
f = open("Test1.txt", "r")  
print(f.read(6))  
f.close()
```

This i



A screenshot of a text editor window titled "Test1.txt". The editor shows a single line of text: "1 This is a test file." with a cursor at the end of the line.

# Reading One Line of a File

```
f = open("Test1.txt", "r")  
print(f.readline())  
f.close()
```

This is a test file.

Test1.txt	
1	This is a test file.
2	This is the second line.

# Reading Lines of A File

```
f = open("Test1.txt", "r")  
for line in f:  
    print(line)
```

This is a test file.

This is the second line.

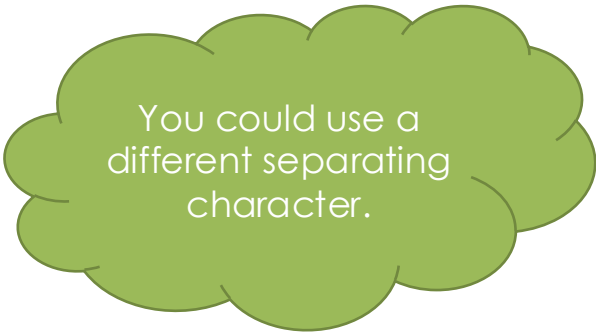
Test1.txt	
1	This is a test file.
2	This is the second line.

# Activity: Reading from a txt File and Creating a Dictionary

```
f = open('Stock.txt','r')
shopItems={}
for line in f:
    item=[]
    for col in line.strip().split(','):
        item.append(col)
    shopItems[int(item[0])]={"Name":item[1],
                           "Price":int(item[2]),
                           "NumberInStock":int(item[3])}

print(shopItems)
```

1	1,Dairy Milk,80,15
2	2,Lucozade Sport,180,10
3	3,Strawberry Laces,100,8



You could use a different separating character.

```
{'1': {'Name': 'Dairy Milk', 'Price': 80, 'Stock': 15},
'2': {'Name': 'Lucozade Sport', 'Price': 180, 'Stock': 10},
'3': {'Name': 'Strawberry Laces', 'Price': 100, 'Stock': 8}}
```

# Activity: Reading from a txt File and Creating a 2D List

```
f = open('Stock.csv', 'r')
shopItems=[]
for line in f:
    item=[]
    for col in line.strip().split(','):
        item.append(col)
    key = int(item[0])
    name = item[1]
    price = int(item[2])
    stock = int(item[3])
    shopItems.append([key,name,price,stock])
f.close()
print(shopItems)
```

1	1,Dairy Milk,80,15
2	2,Lucozade Sport,180,10
3	3,Strawberry Laces,100,8

# Writing to a File

There are two ways to write to a file:

- Append
- Write



# Append to a File

```
Test2.txt
1 This is a test file.
```

```
f = open("Test2.txt", "a")
f.write("This is appended")
f.close()
```

```
Test2.txt
1 This is a test file.This is appended
```

# Write to a File

Test2.txt

```
1 This is a test file.
```

```
f = open("Test2.txt", "w")
f.write("This is written.")
f.close()
```

Test2.txt

```
1 This is written.
```

Notice how write  
overwrites the file!

# Activity: Writing To a txt File From a 2D List

```
shopItems = [[1,"Dairy Milk Plain Chocolate",80,15], [2,"Lucozade Sport",180,10], [3,"Strawberry Laces",30,8]]
```

```
f = open("StockNew.txt", "w")
for row in range(len(shopItems)):
    for col in shopItems[row]:
        text = str(col) + ","
        f.write(text)
    f.write("\n")
```

```
f.close()
```

	A	B	C	D
1	1	Dairy Milk Plain Chocolate	80	15
2	2	Lucozade Sport	180	10
3	3	Strawberry Laces	30	8

# Activity: Writing to a txt File from a Dictionary

```
shopItems = {1:{"Name":"Dairy Milk Plain Chocolate", "Price":80,
"NumberInStock":15},
2:{"Name":"Lucozade Sport", "Price":180, "NumberInStock":10},
3:{"Name":"Strawberry Laces", "Price":30, "NumberInStock":8}}
```

```
f = open("StockNew.txt", "w")
for row in range(len(shopItems)):
    key = row+1
    text = str(key) + "," + shopItems[key]["Name"] + ","
           +str(shopItems[key]["Price"])+","
           +str(shopItems[key]["NumberInStock"]) + "\n"
    f.write(text)
f.close()
```

	A	B	C	D
1	1	Dairy Milk Plain Chocolate	80	15
2	2	Lucozade Sport	180	10
3	3	Strawberry Laces	30	8

# Create A File

```
f = open("myfile.txt", "x")
```



myfile.txt



# Activity: PyShop Reading and Writing to Files

Edit your program so that the stock is read from a csv file when it first loads and then once purchases are complete (e.g. the shop is closed) the stock file is overwritten to be correct.

# Activity: PyShop Reading and Writing to Files Solution

```
def login():  
    ...  
    print("Welcome")  
    load_stock()  
    menu()  
    ...  
  
def menu():  
    ...  
    elif selection==4:  
        print("You selected shut shop")  
        store_stock()  
    ...
```

# Activity: PyShop Reading/Writing to Files (Dictionary) Solution

```
def load_stock():  
    f = open('Stock.csv','r')  
    for line in f:  
        item=[]  
        for col in line.strip().split(','):  
            item.append(col)  
        shopItems[int(item[0])]={"Name":item[1],"Price":int(item[2]),  
                                "NumberInStock":int(item[3])}  
    f.close()
```



# Activity: PyShop Reading/Writing to Files (Dictionary) Solution

```
def store_stock():  
    f = open("Stock.csv", "w")  
    for row in range(len(shopItems)):  
        key = row+1  
        text = str(key) + "," + shopItems[key]["Name"] + "," +  
                str(shopItems[key]["Price"]) + "," +  
                str(shopItems[key]["NumberInStock"]) + "\n"  
        f.write(text)  
    f.close()
```

# Activity: PyShop Reading/Writing to Files (2D List) Solution

```
def load_stock():  
    f = open('Stock.csv', 'r')  
    global shopItems  
    shopItems=[]  
    for line in f:  
        item=[]  
        for col in line.strip().split(','):  
            item.append(col)  
            key = int(item[0])  
            name = item[1]  
            price = int(item[2])  
            stock = int(item[3])  
            shopItems.append([key,name,price,stock])  
    f.close()
```

# Activity: PyShop Reading/Writing to Files (2D List) Solution

```
def store_stock():  
    f = open("Stock.csv", "w")  
    for row in range(len(shopItems)):  
        for col in shopItems[row]:  
            text = str(col) + ","  
            f.write(text)  
        f.write("\n")  
    f.close()
```

# Searching

There's lots of ways to perform searching in Python, we will only cover a few.

# Searching a List

```
shopItems = [  
    [1,"Dairy Milk Plain Chocolate",80,15],  
    [2,"Lucozade Sport",180,10],  
    [3,"Strawberry Laces",30,8]  
]
```

```
search = input("What are you looking for?")  
for item in shopItems:  
    if search in item:  
        print(item[0])
```

What are you looking for?Strawberry Laces  
3

# Searching a Dictionary

```
shopItems = {1:{"Name":"Dairy Milk Plain Chocolate", "Price":80,
"NumberInStock":15}, 2:{"Name":"Lucozade Sport", "Price":180,
"NumberInStock":10}, 3:{"Name":"Strawberry Laces", "Price":30,
"NumberInStock":8}}
```

```
search = input("What are you searching for?")
for key, stock in shopItems.items():
    if stock["Name"] == search:
        print(key)
```

What are you searching for?Strawberry Laces  
3



# Activity: PyShop

## Searching

Edit your PyShop program to include the following search functions:

- Retrieve the key based off the item name
- Retrieve the price based off the item name
- Retrieve the stock number based off the item name
- Retrieve the whole item information based off the key
- Any other search function you think necessary



# Activity: PyShop

## Performing a Calculation

Edit your PyShop program to perform some calculations. We already calculate the total sales so use the following for ideas:

- How many sales have been made
- How much total stock value is left (e.g.  $\text{stock} \times \text{price}$  for each item)





# Activity: Extending Your PyShop

Extend your PyShop to have other functionality. For example:

- A function and menu option to add items to the stock.
- A function to complete multiple purchases in one transaction.
- Your own ideas!

# Interfaces

There are a few types of interfaces:

- Menu Driven
- Command Line
- Graphical

# Menu Driven Interface



# Menu Driven Interface

## Simple Menu

- User offered a simple menu from which to choose an option.
- One menu often leads to a further menu.
- Part of the screen may have an instruction followed by a number of options to choose from.

## Full Screen Menu

- Takes up the entire screen.

## Menu Bar

- Set of options at the top of the screen. When an option is chosen a drop-down menu may be offered.
- Easy to use and the user does not have to remember sets of commands.
- User friendly – can often guess your way around the options.
- Can be irritating if there are too many.

# Command Line Interface

A command-line interface allows the user to interact with the computer by typing in commands. The computer displays a prompt, the user keys in the command and presses enter or return.

# Command Line Interface

- Commands must be typed correctly and in the right order or the command will not work.
- Experienced users who know their commands can work very quickly without having to find their way around menus.
- Command driven programs do not need the memory and processing power of the latest computer and will often run on lower spec machines.

# Graphic User Interface

Graphical user interface is sometimes shortened to GUI. The user chooses an option usually by pointing a mouse at an icon representing that option.

# Graphic User Interface

- Much easier to use for beginners.
- Enable you to easily exchange information between software using cut and paste or drag and drop.
- Use a lot of memory and processing power.
- Can be slower than a command line interface if you are an expert user.
- Can be irritating to experienced users when simple tasks require a number of operations.





# Activity: What Makes a Good Interface?

# A Good Interface

- Be attractive and pleasing to the eye.
- Allow the user to try out different options easily.
- Be easy to use.
- Use suitable colours for key areas.
- Use words that are easy to understand aimed at the type of user.
- Have helpful documentation.

# Tkinter

The **Tkinter** module ("Tk interface") is the standard Python interface to the Tk GUI toolkit.

Tkinter can be used to create a Graphical User Interface for the PyShop.

# Widgets

There are 15 core widgets Tkinter supports:

- Button
- Canvas
- Checkbutton
- Entry
- Frame
- Label
- Listbox
- Menu
- Menubutton
- Message
- Radiobutton
- Scale
- Scrollbar
- Text
- Toplevel

Added in 2.3 Python:

- LabelFrame
- PannedWindow
- Spinbox



# Activity: Which Widget?

# Activity: Which Widget Solution

Button	A simple button, used to execute a command or other operation.
Canvas	Structured graphics. This widget can be used to draw graphs and plots, create graphics editors, and to implement custom widgets.
CheckBox	Represents a variable that can have two distinct values. Clicking the button toggles between the values.
Entry	A text entry field.
Frame	A container widget. The frame can have a border and a background, and is used to group other widgets when creating an application or dialog layout.

# Activity: Which Widget Solution

Label	Displays a text or an image.
Listbox	Displays a list of alternatives. The listbox can be configured to get radiobutton or checklist behavior.
Menu	A menu pane. Used to implement pulldown and popup menus.
Menu Button	A menubutton. Used to implement pulldown menus.
Message	Display a text. Similar to the label widget, but can automatically wrap text to a given width or aspect ratio.
RadioButton	Represents one value of a variable that can have one of many values. Clicking the button sets the variable to that value, and clears all other radiobuttons associated with the same variable.

# Activity: Which Widget Solution

Scale	Allows you to set a numerical value by dragging a “slider”.
Scrollbar	Standard scrollbars for use with canvas, entry, listbox, and text widgets.
Text	Formatted text display. Allows you to display and edit text with various styles and attributes. Also supports embedded images and windows.
Toplevel	A container widget displayed as a separate, top-level window.



# Layout/Geometry Manager

- Pack
- Grid
- Place

# Layout/Geometry Manager - Pack

- Easiest to use
- Don't have to define the precise location, but instead declare the positions relative to each other. The pack command takes care of the details.
- Limited in its possibilities.
- For simple applications it is definitely the manager of choice. For example simple applications like placing a number of widgets side by side, or on top of each other.

# Layout/Geometry Manager - Pack

```
import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack()

w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack()

w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack()

tk.mainloop()
```



# Layout/Geometry Manager - Grid

- Easy to learn and use.
- Produces nicer layouts.
- In many cases the best choice. Pack is sometimes not sufficient for changing details in the layout.
- The Grid geometry manager places the widgets in a 2-dimensional table, which consists of a number of rows and columns.
- The position of a widget is defined by a row and a column number.
- Widgets with the same column number and different row numbers will be above or below each other.
- Correspondingly, widgets with the same row number but different column numbers will be on the same "line" and will be beside of each other, i.e. to the left or the right.

# Layout/Geometry Manager - Grid

```
import tkinter as tk

colours = ['red', 'green', 'orange', 'white', 'yellow', 'blue']
r = 0
for c in colours:
    tk.Label(text=c, relief=tk.RIDGE, width=15).grid(row=r, column=0)
    tk.Entry(bg=c, relief=tk.SUNKEN, width=10).grid(row=r, column=1)
    r = r + 1

tk.mainloop()
```



# Layout/Geometry Manager - Place

- Allows you explicitly set the position and size of a window, either in absolute terms, or relative to another window.
- It can be applied to all standard widgets.

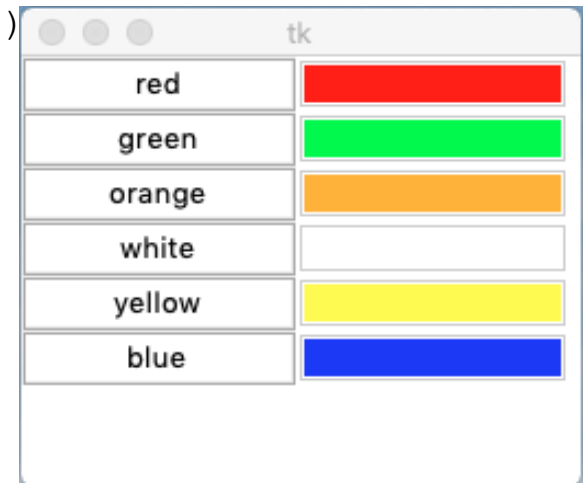
# Layout/Geometry Manager - Place

```
import tkinter as tk

colours = ['red', 'green', 'orange', 'white', 'yellow', 'blue']

x=0
y=0
for c in colours:
    l=tk.Label(text=c, relief=tk.RIDGE, width=15)
    l.place(x=x,y=y,width=125,height=25)
    e=tk.Entry(bg=c, relief=tk.SUNKEN, width=10)
    e.place(x=x+125, y=y,width=125,height=25)
    y=y+25

tk.mainloop()
```



# First Tkinter GUI

```
from tkinter import *
```



Imports the tkinter module which allows us to work with the TK toolkit.

```
root = Tk()
```

```
label = Label(root, text="Hello, world!")
```

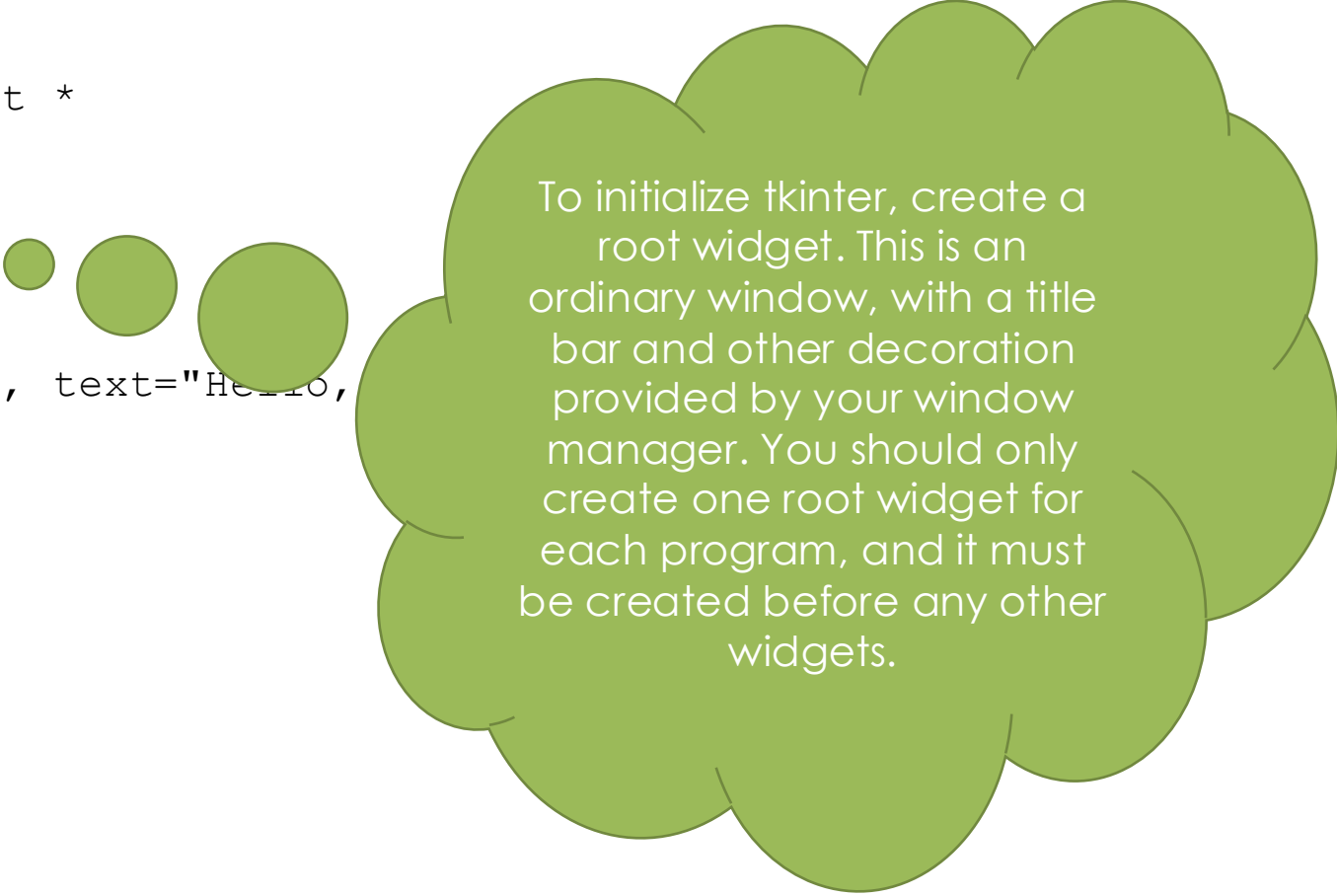
```
label.pack()
```

```
root.mainloop()
```



# First Tkinter GUI

```
from tkinter import *  
  
root = Tk()  
  
label = Label(root, text="Hello,  
label.pack()  
  
root.mainloop()
```



To initialize tkinter, create a root widget. This is an ordinary window, with a title bar and other decoration provided by your window manager. You should only create one root widget for each program, and it must be created before any other widgets.

# First Tkinter GUI


```
from tkinter import *
```

```
root = Tk()
```

```
label = Label(root, text="Hello, world!")
```

```
label.pack()
```

```
root.mainloop()
```



Create a label widget which is a child of the root widget and holds the text "Hello, world!"

# First Tkinter GUI

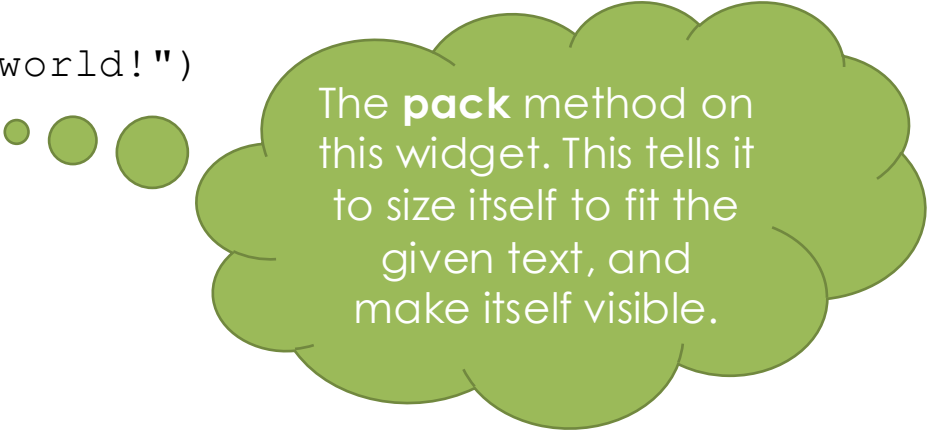
```
from tkinter import *
```

```
root = Tk()
```

```
label = Label(root, text="Hello, world!")
```

```
label.pack()
```

```
root.mainloop()
```



The **pack** method on this widget. This tells it to size itself to fit the given text, and make itself visible.

# First Tkinter GUI

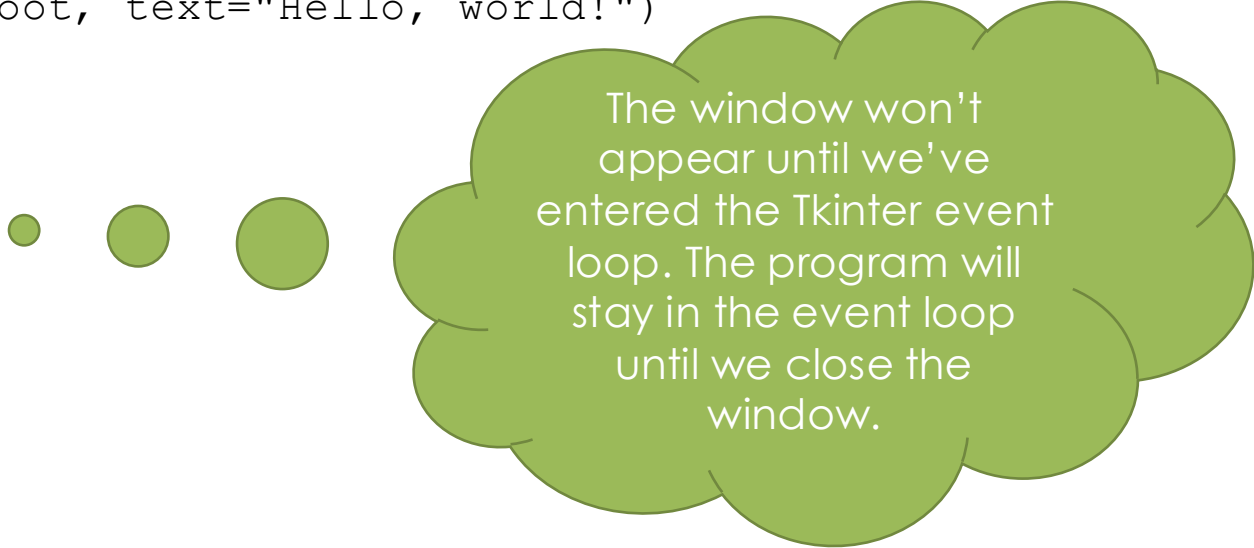
```
from tkinter import *
```

```
root = Tk()
```

```
label = Label(root, text="Hello, world!")
```

```
label.pack()
```

```
root.mainloop()
```



The window won't appear until we've entered the Tkinter event loop. The program will stay in the event loop until we close the window.

# Simple GUI

```
from tkinter import *

def say_hi():
    print("hi there, everyone!")

root = Tk()
frame = Frame(root)
frame.pack()
quitButton = Button(frame, text="QUIT", fg="red", command=root.destroy)
quitButton.pack(side=LEFT)
hiButton = Button(frame, text="Hello", command=say_hi)
hiButton.pack(side=LEFT)
root.mainloop()
```

# Simple GUI

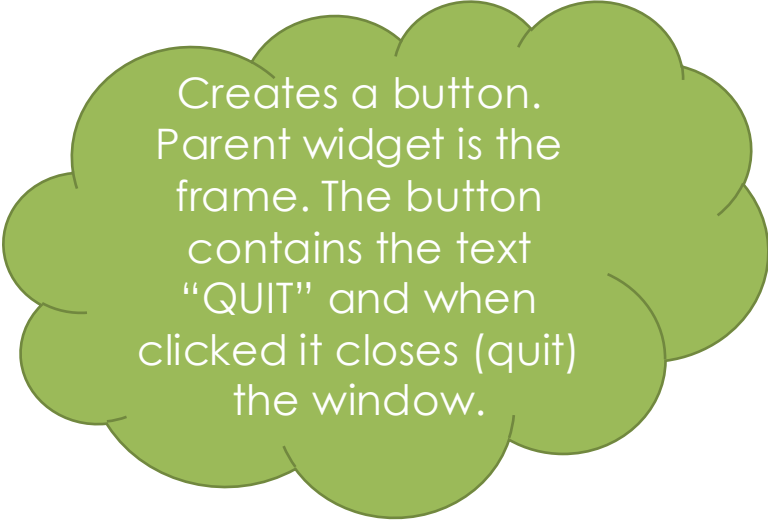
```
from tkinter import *  
  
def say_hi():  
    print("hi there, everyone!")  
  
root = Tk()  
frame = Frame(root)  
frame.pack()  
quitButton = Button(frame, text="QUIT", fg="red", command=root.destroy)  
quitButton.pack(side=LEFT)  
hiButton = Button(frame, text="Hello", command=say_hi)  
hiButton.pack(side=LEFT)  
root.mainloop()
```



Creates a frame.  
The parent is the  
root widget.

# Simple GUI

```
from tkinter import *  
  
def say_hi():  
    print("hi there, everyone!")  
  
root = Tk()  
frame = Frame(root)  
frame.pack()  
quitButton = Button(frame, text="QUIT", fg="red", command=root.destroy)  
quitButton.pack(side=LEFT)  
hiButton = Button(frame, text="Hello", command=say_hi)  
hiButton.pack(side=LEFT)  
root.mainloop()
```

A green thought bubble with a black outline, containing text. Three small green circles lead from the bubble towards the bottom left.

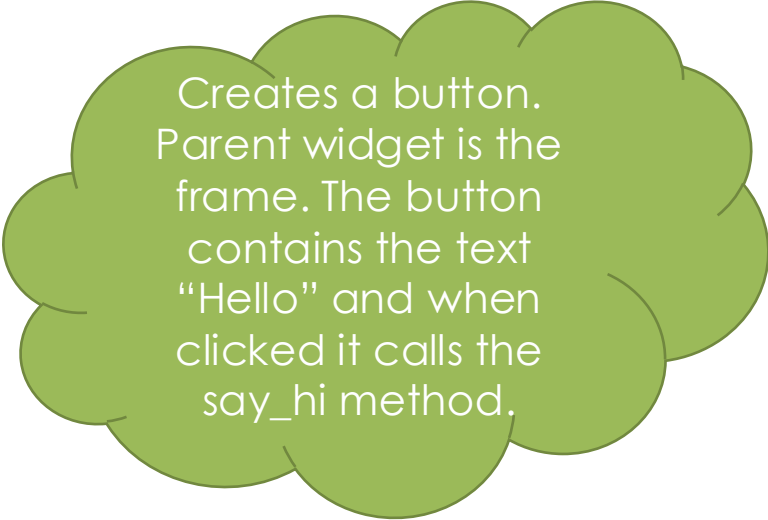
Creates a button.  
Parent widget is the  
frame. The button  
contains the text  
"QUIT" and when  
clicked it closes (quit)  
the window.

# Simple GUI

```
from tkinter import *

def say_hi():
    print("hi there, everyone!")

root = Tk()
frame = Frame(root)
frame.pack()
quitButton = Button(frame, text="QUIT", fg="red", command=root.destroy)
quitButton.pack(side=LEFT)
hiButton = Button(frame, text="Hello", command=say_hi)
hiButton.pack(side=LEFT)
root.mainloop()
```



Creates a button.  
Parent widget is the  
frame. The button  
contains the text  
"Hello" and when  
clicked it calls the  
say\_hi method.



# Simple GUI



Closes Window

hi there, everyone!

# Activity: Simple GUI

```
from tkinter import *

def say_hi():
    print("hi there, everyone!")

root = Tk()
frame = Frame(root)
frame.pack()
quitButton = Button(frame, text="QUIT", fg="red", command=root.destroy)
quitButton.pack(side=LEFT)
hiButton = Button(frame, text="Hello", command=say_hi)
hiButton.pack(side=LEFT)
root.mainloop()
```



# Activity: PyShop GUI

We'll now begin work on the PyShop GUI.

**Create a new Python file to do this.**

Later we will transfer some of the logic across from the current Command Line Interface PyShop we have.



# Activity: PyShop GUI – Login

Create a GUI that looks like the following:

A screenshot of a Tkinter window titled 'tk'. The window has a standard macOS-style title bar with red, yellow, and green buttons. Inside the window, there are two text input fields. The first field is labeled 'username' and the second field is labeled 'password'. Below the 'password' field is a button labeled 'submit'.

Note: The button does **not** have to do anything. Don't use functions.

# Activity: PyShop GUI - Login Solution

```
from tkinter import *
```

```
root = Tk()
```

```
frame = Frame(root)
```

```
frame.grid(row=0,column=0)
```

```
loginFrame=Frame(frame)
```

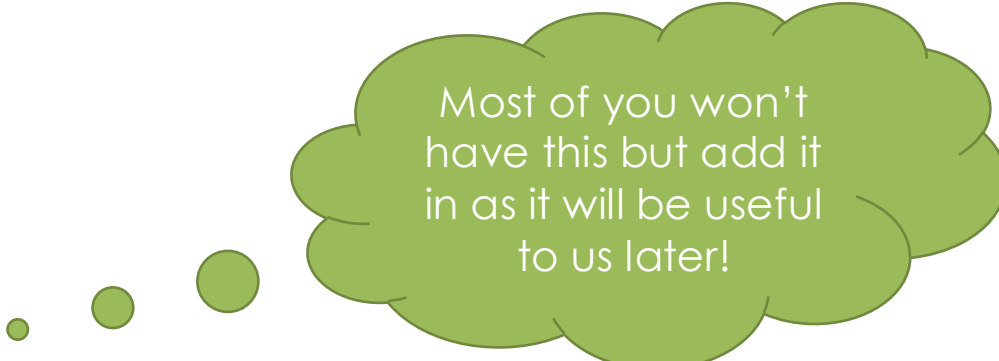
```
loginFrame.grid(row=0,column=0)
```

```
usernameLabel=Label(loginFrame,text="username")
```

```
usernameLabel.grid(row=0,column=0)
```

```
usernameEntry=Entry(loginFrame)
```

```
usernameEntry.grid(row=0,column=1)
```



Most of you won't have this but add it in as it will be useful to us later!

# Activity: PyShop GUI - Login Solution

```
passwordLabel=Label(loginFrame,text="password")  
passwordLabel.grid(row=1,column=0)  
passwordEntry=Entry(loginFrame)  
passwordEntry.grid(row=1,column=1)  
  
submitButton=Button(loginFrame,text="submit")  
submitButton.grid(row=2,column=1)  
root.mainloop()
```

# Button Actions With Parameters

Buttons can perform actions by linking them to functions as explained earlier on.

Parameters can be also passed to functions.

```
submitButton=Button(frame,text="Say Hello",
    command=lambda:say_hello(nameEntry.get()))
```



lambda is key to  
enable these  
types of things to  
work!

# Button Actions With Parameters

```
from tkinter import *

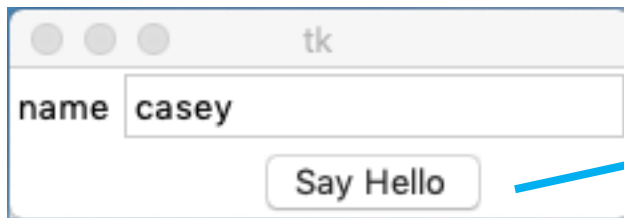
def say_hello(name):
    print("hello",name)

def main():
    nameLabel=Label(frame,text="name")
    nameLabel.grid(row=0,column=0)
    nameEntry=Entry(frame)
    nameEntry.grid(row=0,column=1)
    helloButton=Button(frame,text="Say
        Hello",command=lambda:say_hello(nameEntry.get()))
    helloButton.grid(row=2,column=1)
```



# Button Actions With Parameters

```
root.mainloop()
root = Tk()
frame = Frame(root)
frame.grid(row=0, column=0)
main()
```



hello casey



# Activity: PyShop GUI - Login Button Actions

Edit your login GUI to attempt a login passing in the username and password as parameters from the entry boxes.

Use the below method to check the login. You'll need to put the set up of the GUI in a main method. Call main to start the program.

```
def login(usernameEntry,passwordEntry):  
    if username==usernameEntry and password==passwordEntry:  
        print("Welcome")  
    else:  
        main()
```

# Activity: PyShop GUI - Login Button Actions Solution

```
from tkinter import *
```

```
username="casey"
```

```
password="secret"
```

```
def main():
```

```
    usernameLabel=Label(loginFrame,text="username")
```

```
    usernameLabel.grid(row=0,column=0)
```

```
    usernameEntry=Entry(loginFrame)
```

```
    usernameEntry.grid(row=0,column=1)
```

# Activity: PyShop GUI - Login Button Actions Solution

```
passwordLabel=Label(loginFrame,text="password")
passwordLabel.grid(row=1,column=0)
passwordEntry=Entry(loginFrame)
passwordEntry.grid(row=1,column=1)
submitButton=Button(loginFrame,text="submit",
    command=lambda:login(usernameEntry.get(), passwordEntry.get()))
submitButton.grid(row=2,column=1)

root.mainloop()
```

# Activity: PyShop GUI - Login Button Actions Solution

```
def login(usernameEntry,passwordEntry):  
    if username==usernameEntry and password==passwordEntry:  
        print("Welcome")  
    else:  
        main()
```

```
root = Tk()  
frame = Frame(root)  
frame.grid(row=0,column=0)  
loginFrame=Frame(frame)  
loginFrame.grid(row=0,column=0)  
main()
```



# Activity: PyShop GUI – Items

Edit your GUI to display the items of the shop in a table like structure after the user successfully logs in.

ID	Name	Price	NumberInStock
1	Dairy Milk Plain Chocolate	80	15
2	Lucozade Sport	180	10
3	Strawberry Laces	30	8

1. Add in the items dictionary from your CLI.
2. After `if username==usernameEntry and password==passwordEntry:`
  - a. Destroy the login frame
  - b. Create labels for: id, Name, Price, NumberInStock
  - c. Write an `update_items()` function that iterates through the items and creates labels for each one.

# Activity: PyShop GUI - Items Solution

```
from tkinter import *

username="casey"
password="secret"

shopItems = {
1:{
    "Name":"Dairy Milk Plain Chocolate",
    "Price":80,
    "NumberInStock":15
}, ...

def main():
    ...
```

# Activity: PyShop GUI - Items Solution

```
def login(usernameEntry,passwordEntry):  
    if username==usernameEntry and password==passwordEntry:  
        loginFrame.destroy()  
        idLabel=Label(frame,text="ID")  
        idLabel.grid(row=0,column=0)  
        nameLabel=Label(frame,text="Name")  
        nameLabel.grid(row=0,column=1)  
        priceLabel=Label(frame,text="Price")  
        priceLabel.grid(row=0,column=2)  
        stockLabel=Label(frame,text="NumberInStock")  
        stockLabel.grid(row=0,column=3)  
        update_items()  
    else:  
        main()
```



# Activity: PyShop GUI - Items Solution

```
def update_items():  
    r=1  
    for x in shopItems:  
        label=Label(frame,text=r)  
        label.grid(row=r,column=0)  
        label=Label(frame,text=shopItems[r]["Name"])  
        label.grid(row=r,column=1)  
        label=Label(frame,text=shopItems[r]["Price"])  
        label.grid(row=r,column=2)  
        label=Label(frame,text=shopItems[r]["NumberInStock"])  
        label.grid(row=r,column=3)  
        r=r+1
```

# Activity: PyShop GUI - Items Solution

```
root = Tk()  
frame = Frame(root)  
frame.grid(row=0, column=0)  
loginFrame=Frame(frame)  
loginFrame.grid(row=0, column=0)  
main()
```



# Activity: PyShop GUI - Purchase and Total Sales

1. Add a totalSales variable.
2. Add an entry widget for users to type in the id of the item they wish to purchase.
3. Add a purchase button.
4. Add a label to represent total sales.
5. When the purchase button is clicked:
  - a. One should be taken from the stock (call update\_items to see the change occur onscreen)
  - b. The total sales should be updated. (create an update\_sales function)

# Message Box

You can get a message box to pop up using the following:

```
from tkinter import *  
from tkinter import messagebox  
  
def clicked():  
    messagebox.showwarning("Warning", "Out of stock")  
  
root = Tk()  
frame = Frame(root)  
frame.grid(row=0, column=0)  
button=Button(frame, text="message box", command=lambda:clicked())  
button.grid(row=0, column=0)  
root.mainloop()
```

# PyShop Validation Issues

Issues:

- Stock can go below 0.
- What happens if you enter a number that is not a correct id?
- What happens if you enter a character that's not an integer?



# Activity: PyShop GUI – Validation

1. Add some code to check the stock levels before a purchase is completed.
  - a. If there is 0 stock left then a warning message should pop up.
2. Add code to capture any errors that could occur in the purchase method.

# Sports Management System

The Welsh government want to encourage more children into participating in sports outside of school. They are therefore investing in a system that sports teams can use to manage their teams. They hope that they will be able to collect the data from each team at the end of the year in order to track how many children are participating.

Managers of Welsh sports teams have been contacted to ask what features they would like in such systems. As there are a number of different sports teams available a blanket default system is wanted to begin with before additional features are added later on.

# Sports Management System

The system is expected to manage the information held on players such as name, DOB, and address. Each player is also allocated to a team. E.g. Team A, Team B, Team C.

The manager of the club is expected to log in to retrieve player information. They can also allocate players to teams. Teams can hold up to 15 players.

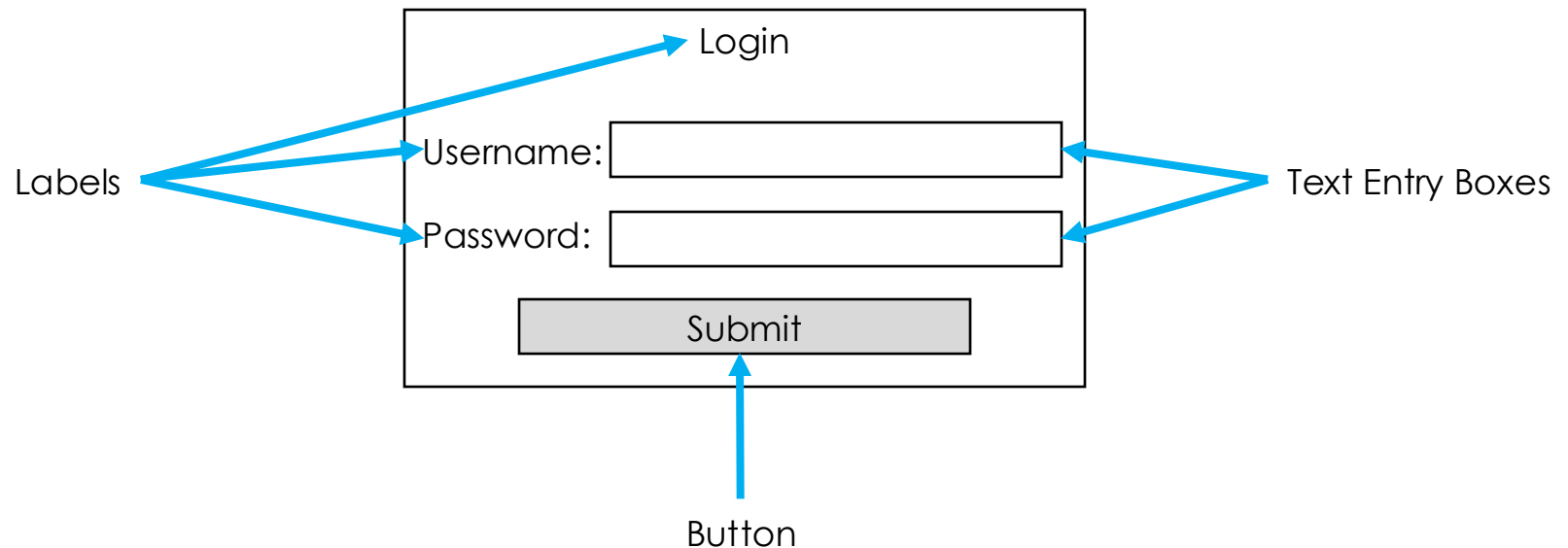
The system must also manage the subs payments. Each training session (held weekly) is £3, however if players attend 3 weeks in a row they receive a £1.50 discount on their 4th session, as an incentive to return.



# Design

It is very important to design a program before writing it. This allows you to make decisions in advance and saves you from wasting time.

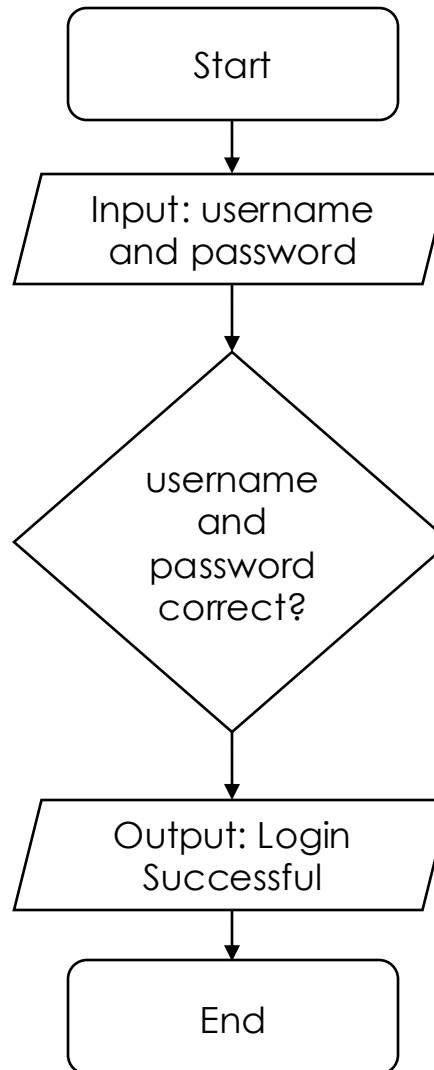
# User Interface Design



# Selecting Data Structures

You need to select appropriate data structures to hold information. For example would you use a string to hold a list of items or would you use a list?

# Flowcharts



# Pseudocode

Set total to zero

Set grade counter to one

While grade counter is less than or equal to ten

Input the next grade

Add the grade to the total

Set the class average to the total divided by ten

Print the class average

# Activity: Design

Develop a design for the Sports Management System.

# Activity: Implementing The Sports Management System

## **Effectiveness of Solution [15 marks]**

You need to make sure that the finished application:

- is functional and fulfils all the requirements of the Welsh Government
- has an interface that is easy to use
- is modular and makes efficient use of resources
- has authentication routines
- is reliable and robust.

## **Technical Quality [20 marks]**

- is self-documenting and well structured
- uses a consistent style throughout including indentation and use of white space
- uses meaningful identifiers and appropriate constants
- uses local variables to minimise the use of global variables
- has validation routines and can handle errors such as division by zero
- has informed annotation to demonstrate your understanding of the solution.

# Testing A System

All systems/programs need to be tested before they are released! Imagine an aeroplane system, what could happen if it wasn't tested before being implemented on a plane?!

There are many different methods of testing, they can be in depth or quite simple.

For our tests we will perform an action and then compare the actual result to the expected/desired result.



# Testing A System

Test Name	Input	Expected Output	Actual Output	Pass/Fail
Login test with valid information	Username = "username" Password = "password"	Login success	Login success	Pass
Login test with invalid information	Username = "user" Password = "1234"	Login failure	Login success	Fail
Testing the addition of the system	Variable a = 1 Variable b = 2	$1+2 = 3$	$1+2 = 3$	Pass

# Activity: Testing The Sports Management System

Test the system you have developed so far ensuring it behaves as expected. If you discover an issue with your program try to debug the error and fix the program.