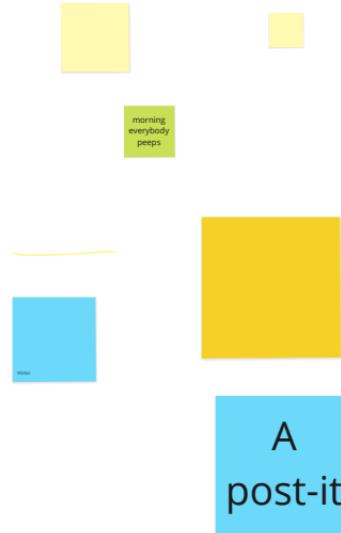


Add some  
post its!

Press N to  
create your  
own sticky  
notes



Draw  
something

Press P to  
create your art



Write  
something

Press T to  
express your  
thoughts

This is text

howdy

**Name?**

**Where you work?**

**What you do?**

**Programming experience?**

**Sports?**

**Hobbies?**

**Interests?**

## *Introduce yourself*

Rich W  
Lidl GB  
IT Support  
Some experience  
No sport

Tom  
Lidl GB  
IT Consultant -  
Retail Solutions  
None  
Home automation

Elliott  
Lidl  
Server & Client  
Analyst  
Some Java at uni  
Swimming Running

Pete  
QA  
Trainer  
Decades  
Stop Motion  
Animation

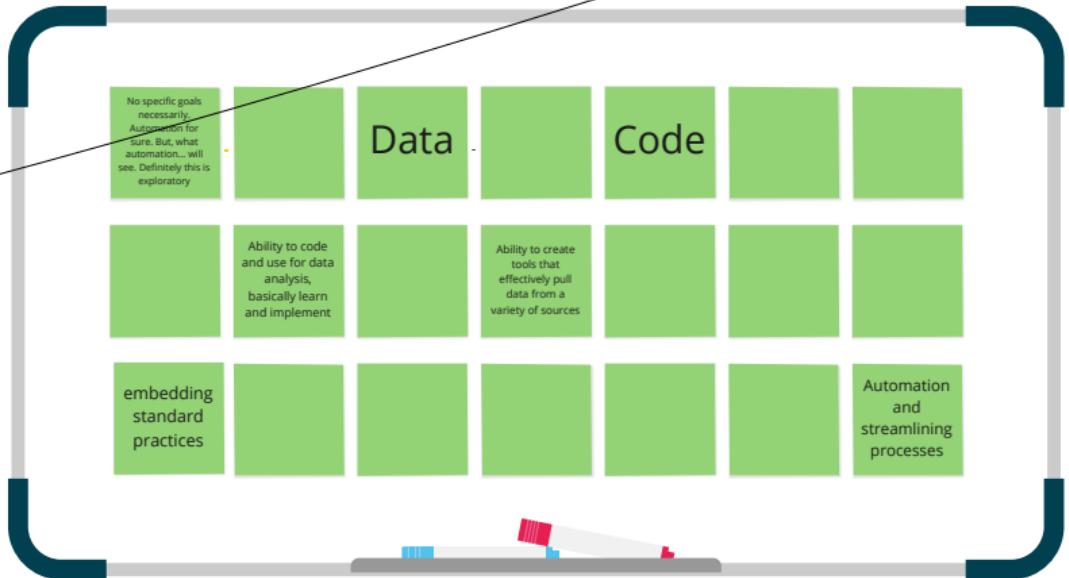
Jay Tumati  
GBG  
Operational Performance  
Engineer  
Came from QA  
Background, Not much of  
Prgm Exp  
Badminton

Dav aka Andrew Davenport  
LBG  
work with python now and  
have previously been a  
cobol programmer  
Rugby League, RC crawling,  
golf

Shaun  
Hunter  
work at  
LBG as SM

Dylan  
Lidl GB  
IT Consultant  
Computer Science at  
Uni 6 years ago  
Football, music,  
travelling

## Any specific goals for the course?



## And your TEAMS for the QUEST!

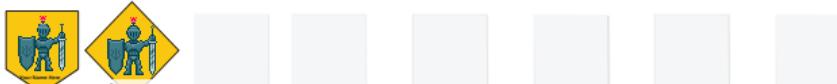
### Excalibur



### Castle Greyskull



### Brave Knight



### European Swallows



### Peasants



### Brave Horses



Group Learners into BreakoutGroups: Shoe/Pram/Swallow/Coconuts

HOW MANY Monty Python References -Count them during the course!!

## Questions & Comments

Questions for tutor and open to all students who go here..

Kanban

To do | 10

In progress | 0

Done | 0

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

## Code shared by tutor

<Code shared by tutor will go here...

Link for Project Starter Files:

[QACS-TL/Python\\_TL\\_Lab\\_Starter\\_Files.github.com](https://QACS-TL/Python_TL_Lab_Starter_Files.github.com)

Link for Demo Code:

[QACS-TL/Python\\_TL\\_Demos.github.com](https://QACS-TL/Python_TL_Demos.github.com)

Link for Lab Solutions:

[QACS-TL/Python\\_TL\\_Lab\\_Sols.github.com](https://QACS-TL/Python_TL_Lab_Sols.github.com)

## Code shared by learner

<Code shared by learner will go here...

### Fun Data Menu

- 1. Display Online Public Weather Data to Browser.
- 2. Display Online Public UK Police Data to Screen.
- 3. Display/Search top Movies from IMDb.

Enter an option ( 1-3, [qQ= quit] ):

#### Weather Data

- 1. Get online weather data for city, country
- 2. Display weather data to web browser
- 3. Save weather data to local database
- 4. Query all news in local database
- 5. Delete news in local database
- 6. Delete all news in local database
- 7. Drop local database table

Enter an option ( 1-7, [qQ= quit] ):



#### Welcome to PDCA - the Police Data Crime App

##### Police Data

- 1. List of Police Forces
- 2. Crimes by location
- 3. Stop and Search by Location

Please choose an option ( 1-3 [qQ= quit] ):



#### Movie Menu

- 1. Get online movie ranking from IMDb
- 2. Display top ranking movies
- 3. Search for movie

Enter option ( 1-3, [qQ= quit] ):



```
#!/bin/python
# Name: menu.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: Fun demo for QA2.0 Python LIVE
```

Docstring: A fun program to allow users to choose from an interesting and eclectic list of options including getting the latest weather or crime reports from an area, getting the latest top 250 movies and interacting with SQLite databases.

And a little 8\_Object\_Oriented\_Programming for extra fun!

```
import sys
import police_data
import weather_data
import movies

menu = """
Fun Data Menu
-----
1. Display Online Public Weather Data to Browser.
2. Display Online Public UK Police Data to Screen.
3. Display/Search top Movies from IMDb.

"""

def top_level_menu():
    """ Top Level Menu function for displaying crime, weather and movie data """
    while True:
        print (menu)
        option = input ( "Enter an option (1-3, [qQ=quit]): " )
        if option == "1":
            weather_data.weather_menu()
        elif option == "2":
            police_data.police_menu()
        elif option == "3":
            movies.menu()
        elif option.lower () == "q":
            print ( "Quitting." )
            break
        else:
            print ( "Invalid option" )
    return None

def main():
    """ Program can be run directly """
    top_level_menu()
    return None

if __name__ == "__main__":
    main()
    sys.exit ( 0 )
```







```
#!/usr/bin/python
# Name: police_data.py
# Author: QAA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will allow the user to retrieve publicly
# available Police Forces, crime and stop and search information for
# specific areas using simple HTTP GET requests.
# Decoding: This program/module will.
# imports
import webbrowser
import requests
import sys
import json

app_name = "Welcome to PDCA - the Police Data Crime App"
title = " "
Police Data
1. List of Police Forces
2. Crimes by location
3. Stop and Search by Location
base_url = "http://data.police.uk/api"

def getPoliceForces ():
    """ Display List of Police Forces """
    try:
        full_url = base_url + "forces"
        force_data = requests . get (full_url) . json () # HTTP-GET at page
        for force in force_data . json () :
            print (f" {force [ 'id' ]} {force [ 'name' ]} {force [ 'name' ] >200} ")
        print (0)
    except Exception as err:
        print (f"Error, cannot retrieve info {err} ")
    return None

def getCrimeByLoc ():
    """ Display Police Crime data for a specific lat-long """
    data = input ("Enter a date (YYYY-MM) ")
    location = input ("Enter a Lat/Long (eg. 51.506-0.075 (London Bridge)) ")
    if not location :
        lat = "51.506"
        lon = "-0.075"
    else :
        lat , lon = location . split (",")
    try:
        full_url = base_url + "crimes-at-location?lat=" + lat + "&long=" + lon + "&date=" + data
        crime_data = requests . get (full_url) . json ()
        for crime in crime_data . json () :
            print (f" {crime [ 'category' ]} {crime [ 'location_type' ]} {crime [ 'location_status' ]} {crime [ 'date' ]} {crime [ 'location' ]} {crime [ 'id' ]} {crime [ 'name' ]} ")
        print (0)
    except Exception as err:
        print (f"Error, {err} ")
    return None

def getStopByArea ():
    """ Display Police Stop Data for a specific lat-long """
    data = input ("Enter a date (YYYY-MM) ")
    location = input ("Enter a Lat/Long (eg. 51.506-0.075 (London Bridge)) ")
    if not location :
        lat = "51.506"
        lon = "-0.075"
    else :
        lat , lon = location . split (",")
    try:
        full_url = base_url + "stops-at-area?lat=" + lat + "&long=" + lon + "&date=" + data
        stops_data = requests . get (full_url) . json ()
        for stops in stops_data . json () :
            print (f" {stops [ 'subject_of_search' ]} {stops [ 'gender' ]} {stops [ 'age_group' ]} {stops [ 'type' ]} {stops [ 'location' ]} {stops [ 'id' ]} {stops [ 'name' ]} ")
        print (0)
    except Exception as err:
        print (f"Error, {err} ")
    return None
finally:
    print (0)

return None

def police_main ():
    """ Main for Police Crime Data """
    while True:
        print (app_name)
        print (f" {1} {2} {3} {4} {5} {6} {7} {8} {9} {10} {11} {12} {13} {14} {15} {16} {17} {18} {19} {20} {21} {22} {23} {24} {25} {26} {27} {28} {29} {30} {31} {32} {33} {34} {35} {36} {37} {38} {39} {40} {41} {42} {43} {44} {45} {46} {47} {48} {49} {50} {51} {52} {53} {54} {55} {56} {57} {58} {59} {60} {61} {62} {63} {64} {65} {66} {67} {68} {69} {70} {71} {72} {73} {74} {75} {76} {77} {78} {79} {80} {81} {82} {83} {84} {85} {86} {87} {88} {89} {90} {91} {92} {93} {94} {95} {96} {97} {98} {99} {100} {101} {102} {103} {104} {105} {106} {107} {108} {109} {110} {111} {112} {113} {114} {115} {116} {117} {118} {119} {120} {121} {122} {123} {124} {125} {126} {127} {128} {129} {130} {131} {132} {133} {134} {135} {136} {137} {138} {139} {140} {141} {142} {143} {144} {145} {146} {147} {148} {149} {150} {151} {152} {153} {154} {155} {156} {157} {158} {159} {160} {161} {162} {163} {164} {165} {166} {167} {168} {169} {170} {171} {172} {173} {174} {175} {176} {177} {178} {179} {180} {181} {182} {183} {184} {185} {186} {187} {188} {189} {190} {191} {192} {193} {194} {195} {196} {197} {198} {199} {200} {201} {202} {203} {204} {205} {206} {207} {208} {209} {210} {211} {212} {213} {214} {215} {216} {217} {218} {219} {220} {221} {222} {223} {224} {225} {226} {227} {228} {229} {230} {231} {232} {233} {234} {235} {236} {237} {238} {239} {240} {241} {242} {243} {244} {245} {246} {247} {248} {249} {250} {251} {252} {253} {254} {255} {256} {257} {258} {259} {260} {261} {262} {263} {264} {265} {266} {267} {268} {269} {270} {271} {272} {273} {274} {275} {276} {277} {278} {279} {280} {281} {282} {283} {284} {285} {286} {287} {288} {289} {290} {291} {292} {293} {294} {295} {296} {297} {298} {299} {300} {301} {302} {303} {304} {305} {306} {307} {308} {309} {310} {311} {312} {313} {314} {315} {316} {317} {318} {319} {320} {321} {322} {323} {324} {325} {326} {327} {328} {329} {330} {331} {332} {333} {334} {335} {336} {337} {338} {339} {340} {341} {342} {343} {344} {345} {346} {347} {348} {349} {350} {351} {352} {353} {354} {355} {356} {357} {358} {359} {360} {361} {362} {363} {364} {365} {366} {367} {368} {369} {370} {371} {372} {373} {374} {375} {376} {377} {378} {379} {380} {381} {382} {383} {384} {385} {386} {387} {388} {389} {390} {391} {392} {393} {394} {395} {396} {397} {398} {399} {400} {401} {402} {403} {404} {405} {406} {407} {408} {409} {410} {411} {412} {413} {414} {415} {416} {417} {418} {419} {420} {421} {422} {423} {424} {425} {426} {427} {428} {429} {430} {431} {432} {433} {434} {435} {436} {437} {438} {439} {440} {441} {442} {443} {444} {445} {446} {447} {448} {449} {450} {451} {452} {453} {454} {455} {456} {457} {458} {459} {460} {461} {462} {463} {464} {465} {466} {467} {468} {469} {470} {471} {472} {473} {474} {475} {476} {477} {478} {479} {480} {481} {482} {483} {484} {485} {486} {487} {488} {489} {490} {491} {492} {493} {494} {495} {496} {497} {498} {499} {500} {501} {502} {503} {504} {505} {506} {507} {508} {509} {510} {511} {512} {513} {514} {515} {516} {517} {518} {519} {520} {521} {522} {523} {524} {525} {526} {527} {528} {529} {530} {531} {532} {533} {534} {535} {536} {537} {538} {539} {540} {541} {542} {543} {544} {545} {546} {547} {548} {549} {550} {551} {552} {553} {554} {555} {556} {557} {558} {559} {550} {551} {552} {553} {554} {555} {556} {557} {558} {559} {560} {561} {562} {563} {564} {565} {566} {567} {568} {569} {570} {571} {572} {573} {574} {575} {576} {577} {578} {579} {580} {581} {582} {583} {584} {585} {586} {587} {588} {589} {580} {581} {582} {583} {584} {585} {586} {587} {588} {589} {590} {591} {592} {593} {594} {595} {596} {597} {598} {599} {590} {591} {592} {593} {594} {595} {596} {597} {598} {599} {600} {601} {602} {603} {604} {605} {606} {607} {608} {609} {600} {601} {602} {603} {604} {605} {606} {607} {608} {609} {610} {611} {612} {613} {614} {615} {616} {617} {618} {619} {610} {611} {612} {613} {614} {615} {616} {617} {618} {619} {620} {621} {622} {623} {624} {625} {626} {627} {628} {629} {620} {621} {622} {623} {624} {625} {626} {627} {628} {629} {630} {631} {632} {633} {634} {635} {636} {637} {638} {639} {630} {631} {632} {633} {634} {635} {636} {637} {638} {639} {640} {641} {642} {643} {644} {645} {646} {647} {648} {649} {640} {641} {642} {643} {644} {645} {646} {647} {648} {649} {650} {651} {652} {653} {654} {655} {656} {657} {658} {659} {650} {651} {652} {653} {654} {655} {656} {657} {658} {659} {660} {661} {662} {663} {664} {665} {666} {667} {668} {669} {660} {661} {662} {663} {664} {665} {666} {667} {668} {669} {670} {671} {672} {673} {674} {675} {676} {677} {678} {679} {670} {671} {672} {673} {674} {675} {676} {677} {678} {679} {680} {681} {682} {683} {684} {685} {686} {687} {688} {689} {680} {681} {682} {683} {684} {685} {686} {687} {688} {689} {690} {691} {692} {693} {694} {695} {696} {697} {698} {699} {690} {691} {692} {693} {694} {695} {696} {697} {698} {699} {700} {701} {702} {703} {704} {705} {706} {707} {708} {709} {700} {701} {702} {703} {704} {705} {706} {707} {708} {709} {710} {711} {712} {713} {714} {715} {716} {717} {718} {719} {710} {711} {712} {713} {714} {715} {716} {717} {718} {719} {720} {721} {722} {723} {724} {725} {726} {727} {728} {729} {720} {721} {722} {723} {724} {725} {726} {727} {728} {729} {730} {731} {732} {733} {734} {735} {736} {737} {738} {739} {730} {731} {732} {733} {734} {735} {736} {737} {738} {739} {740} {741} {742} {743} {744} {745} {746} {747} {748} {749} {740} {741} {742} {743} {744} {745} {746} {747} {748} {749} {750} {751} {752} {753} {754} {755} {756} {757} {758} {759} {750} {751} {752} {753} {754} {755} {756} {757} {758} {759} {760} {761} {762} {763} {764} {765} {766} {767} {768} {769} {760} {761} {762} {763} {764} {765} {766} {767} {768} {769} {770} {771} {772} {773} {774} {775} {776} {777} {778} {779} {770} {771} {772} {773} {774} {775} {776} {777} {778} {779} {780} {781} {782} {783} {784} {785} {786} {787} {788} {789} {780} {781} {782} {783} {784} {785} {786} {787} {788} {789} {790} {791} {792} {793} {794} {795} {796} {797} {798} {799} {790} {791} {792} {793} {794} {795} {796} {797} {798} {799} {800} {801} {802} {803} {804} {805} {806} {807} {808} {809} {800} {801} {802} {803} {804} {805} {806} {807} {808} {809} {810} {811} {812} {813} {814} {815} {816} {817} {818} {819} {810} {811} {812} {813} {814} {815} {816} {817} {818} {819} {820} {821} {822} {823} {824} {825} {826} {827} {828} {829} {820} {821} {822} {823} {824} {825} {826} {827} {828} {829} {830} {831} {832} {833} {834} {835} {836} {837} {838} {839} {830} {831} {832} {833} {834} {835} {836} {837} {838} {839} {840} {841} {842} {843} {844} {845} {846} {847} {848} {849} {840} {841} {842} {843} {844} {845} {846} {847} {848} {849} {850} {851} {852} {853} {854} {855} {856} {857} {858} {859} {850} {851} {852} {853} {854} {855} {856} {857} {858} {859} {860} {861} {862} {863} {864} {865} {866} {867} {868} {869} {860} {861} {862} {863} {864} {865} {866} {867} {868} {869} {870} {871} {872} {873} {874} {875} {876} {877} {878} {879} {870} {871} {872} {873} {874} {875} {876} {877} {878} {879} {880} {881} {882} {883} {884} {885} {886} {887} {888} {889} {880} {881} {882} {883} {884} {885} {886} {887} {888} {889} {890} {891} {892} {893} {894} {895} {896} {897} {898} {899} {890} {891} {892} {893} {894} {895} {896} {897} {898} {899} {900} {901} {902} {903} {904} {905} {906} {907} {908} {909} {900} {901} {902} {903} {904} {905} {906} {907} {908} {909} {910} {911} {912} {913} {914} {915} {916} {917} {918} {919} {910} {911} {912} {913} {914} {915} {916} {917} {918} {919} {920} {921} {922} {923} {924} {925} {926} {927} {928} {929} {920} {921} {922} {923} {924} {925} {926} {927} {928} {929} {930} {931} {932} {933} {934} {935} {936} {937} {938} {939} {930} {931} {932} {933} {934} {935} {936} {937} {938} {939} {940} {941} {942} {943} {944} {945} {946} {947} {948} {949} {940} {941} {942} {943} {944} {945} {946} {947} {948} {949} {950} {951} {952} {953} {954} {955} {956} {957} {958} {959} {950} {951} {952} {953} {954} {955} {956} {957} {958} {959} {960} {961} {962} {963} {964} {965} {966} {967} {968} {969} {960} {961} {962} {963} {964} {965} {966} {967} {968} {969} {970} {971} {972} {973} {974} {975} {976} {977} {978} {979} {970} {971} {972} {973} {974} {975} {976} {977} {978} {979} {980} {981} {982} {983} {984} {985} {986} {987} {988} {989} {980} {981} {982} {983} {984} {985} {986} {987} {988} {989} {990} {991} {992} {993} {994} {995} {996} {997} {998} {999} {990} {991} {992} {993} {994} {995} {996} {997} {998} {999} {1000} {1001} {1002} {1003} {1004} {1005} {1006} {1007} {1008} {1009} {1000} {1001} {1002} {1003} {1004} {1005} {1006} {1007} {1008} {1009} {1010} {1011} {1012} {1013} {1014} {1015} {1016} {1017} {1018} {1019} {1010} {1011} {1012} {1013} {1014} {1015} {1016} {1017} {1018} {1019} {1020} {1021} {1022} {1023} {1024} {1025} {1026} {1027} {1028} {1029} {1020} {1021} {1022} {1023} {1024} {1025} {1026} {1027} {1028} {1029} {1030} {1031} {1032} {1033} {1034} {1035} {1036} {1037} {1038} {1039} {1030} {1031} {1032} {1033} {1034} {1035} {1036} {1037} {1038} {1039} {1040} {1041} {1042} {1043} {1044} {1045} {1046} {1047} {1048} {1049} {1040} {1041} {1042} {1043} {1044} {1045} {1046} {1047} {1048} {1049} {1050} {1051} {1052} {1053} {1054} {1055} {1056} {1057} {1058} {1059} {1050} {1051} {1052} {1053} {1054} {1055} {1056} {1057} {1058} {1059} {1060} {1061} {1062} {1063} {1064} {1065} {1066} {1067} {1068} {1069} {1060} {1061} {1062} {1063} {1064} {1065} {1066} {1067} {1068} {1069} {1070} {1071} {1072} {1073} {1074} {1075} {1076} {1077} {1078} {1079} {1070} {1071} {1072} {1073} {1074} {1075} {1076} {1077} {1078} {1079} {1080} {1081} {1082} {1083} {1084} {1085} {1086} {1087} {1088} {1089} {1080} {1081} {1082} {1083} {1084} {1085} {1086} {1087} {1088} {1089} {1090} {1091} {1092} {1093} {1094} {1095} {1096} {1097} {1098} {1099} {1090} {1091} {1092} {1093} {1094} {1095} {1096} {1097} {1098} {1099} {1100} {1101} {1102} {1103} {1104} {1105} {1106} {1107} {1108} {1109} {1100} {1101} {1102} {1103} {1104} {1105} {1106} {1107} {1108} {1109} {1110} {1111} {1112} {1113} {1114} {1115} {1116} {1117} {1118} {1119} {1110} {1111} {1112} {1113} {1114} {1115} {1116} {1117} {1118} {1119} {1120} {1121} {1122} {1123} {1124} {1125} {1126} {1127} {1128} {1129} {1120} {1121} {1122} {1123} {1124} {1125} {1126} {1127} {1128} {1129} {1130} {1131} {1132} {1133} {1134} {1135} {1136} {1137} {1138} {1139} {1130} {1131} {1132} {1133} {1134} {1135} {1136} {1137} {1138} {1139} {1140} {1141} {1142} {1143} {1144} {1145} {1146} {1147} {1148} {1149} {1140} {1141} {1142} {1143} {1144} {1145} {1146} {1147} {1148} {1149} {1150} {1151} {1152} {1153} {1154} {1155} {1156} {1157} {1158} {1159} {1150} {1151} {1152} {1153} {1154} {1155} {1156} {1157} {1158} {1159} {1160} {1161} {1162} {1163} {1164} {1165} {1166} {1167} {1168} {1169} {1160} {1161} {1162} {1163} {1164} {1165} {1166} {1167} {1168} {1169} {1170} {1171} {1172} {1173} {1174} {1175} {1176} {1177} {1178} {1179} {1170} {1171} {1172} {1173} {1174} {1175} {1176} {1177} {1178} {1179} {1180} {1181} {1182} {1183} {1184} {1185} {1186} {1187} {1188} {1189} {1180} {1181} {1182} {1183} {1184} {1185} {1186} {1187} {1188} {1189} {1190} {1191} {1192} {1193} {1194} {1195} {1196} {1197} {1198} {1199} {1190} {1191} {1192} {1193} {1194} {1195} {1196} {1197} {1198} {1199} {1200} {1201} {1202} {1203} {1204} {1205} {1206} {1207} {1208} {1209} {1200} {1201} {1202} {1203} {1204} {1205} {1206} {1207} {1208} {1209} {1210} {1211} {1212} {1213} {1214} {1215} {1216} {1217} {1218} {1219} {1210} {1211} {1212} {1213} {1214} {1215} {1216} {1217} {1218} {1219} {1220} {1221} {1222} {1223} {1224} {1225} {1226} {1227} {1228} {1229} {1220} {1221} {1222} {1223} {1224} {1225} {1226} {1227} {1228} {1229} {1230} {1231} {1232} {1233} {1234} {1235} {1236} {1237} {1238} {1239} {1230} {1231} {1232} {1233} {1234} {1235} {1236} {1237} {1238} {1239} {1240} {1241} {1242} {1243} {1244} {1245} {1246} {1247} {1248} {1249} {1240} {1241} {1242} {1243} {1244} {1245} {1246} {1247} {1248} {1249} {1250} {1251} {1252} {1253} {1254} {1255} {1256} {1257} {1258} {1259} {1250} {1251} {1252} {1253} {1254} {1255} {1256} {1257} {1258} {1259} {1260} {1261} {1262} {1263} {1264} {1265} {1266} {1267} {1268} {1269} {1260} {1261} {1262} {1263} {1264} {1265} {1266} {1267} {1268} {1269} {1270} {1271} {1272} {1273} {1274} {1275} {1276} {1277} {1278} {1279} {1270} {1271} {1272} {1273} {1274} {1275} {1276} {1277} {1278} {1279} {1280} {1281} {1282} {1283} {1284} {1285} {1286} {1287} {1288} {1289} {1280} {1281} {1282} {1283} {1284} {1285} {1286} {1287} {1288} {1289} {1290} {1291} {1292} {1293} {1294} {1295} {1296} {1297} {1298} {1299} {1290} {1291} {1292} {1293} {1294} {1295} {1296} {1297} {1298} {1299} {1300} {1301} {1302} {1303} {1304} {1305} {1306} {1307} {1308} {1309} {1300} {1301} {1302} {1303} {1304} {1305} {1306} {1307} {1308} {1309} {1310} {1311} {1312} {1313} {1314} {1315} {1316} {1317} {1318} {1319} {1310} {1311} {1312} {1313} {1314} {1315} {1316} {1317} {1318} {1319} {1320} {1321} {1322} {1323} {1324} {1325} {1326} {1327} {1328} {1329} {1320} {1321} {1322} {1323} {1324} {1325} {1326} {1327} {1328} {1329} {1330} {1331} {1332} {1333} {1334} {1335} {1336} {1337} {1338} {1339} {1330} {1331} {1332} {1333} {1334} {1335} {1336} {1337} {1338} {1339} {1340} {1341} {1342} {1343} {1344} {1345} {1346} {1347} {1348} {1349} {1340} {1341} {1342} {1343} {1344} {1345} {1346} {1347} {1348} {1349} {1350} {1351} {1352} {1353} {1354} {1355} {1356} {1357} {1358} {1359} {1350} {13
```



```
# file: weather_db.py
# Name: weather_db.py
# Author: QAZBO, Donald Cramton
# Revision: v1.0
# Description: This module defines a Weather class for storing data
# in a sqlite3 database.
#
# This module describes a class of Weather objects which can create drop tables
# and insert data into a weather database.
#
# import sys
# import sqlite3
# import json
#
# class Weather_db():
#     # Class variable storing location of DB
#     # Could store this in an external file or environment variable.
#     DB_LOC = "C:\lab\weather_db.sqlite"
#
#     def __init__(self):
#         self._db_conn = sqlite3.connect(self.DB_LOC)
#         self._cur = self._db_conn.cursor()
#     except Exception as ex:
#         raise ValueError
#     # No return as not explicitly called.
#
#     def create_table(self, table_name = "weather"):
#         self._cur.execute(f"CREATE TABLE IF NOT EXISTS {table_name} (id INTEGER PRIMARY KEY, city VARCHAR(30), country VARCHAR(30), date DATE DEFAULT CURRENT_DATE, description NVARCHAR(50), temp FLOAT, humidity FLOAT, wind_speed FLOAT, wind_direction NVARCHAR(2))")
#         return None
#
#     def drop_table(self, table_name = "weather"):
#         """Drop table if exists"""
#         option = input("Are you sure you want to drop the table_name? (y/n): ")
#         if option.lower() == "y":
#             self._cur.execute(f"DROP TABLE IF EXISTS {table_name}")
#         return None
#
#     def query_all(self, table_name = "weather"):
#         """Query and return all rows in formatted as output"""
#         sql = f"SELECT * FROM {table_name}"
#         self._cur.execute(sql)
#         rows = self._cur.fetchall()
#         for row in rows:
#             print(row)
#         except Exception as ex:
#             print(f"Error accessing {table_name} ({ex})")
#         return None
#
#     def insert_row(self, table_name = "weather", **kwargs):
#         """Insert new row into weather db table"""
#         weather = { "id" : None, "city" : None, "country" : None, "date" : "DEFAULT", "description" : "None", "temp" : None, "humidity" : None, "wind_speed" : None, "wind_direction" : None }
#         weather.update(kwargs)
#         print(weather)
#         sql = f"INSERT INTO {table_name} VALUES ({weather['id']}, {weather['city']}, {weather['country']}, {weather['date']}, {weather['description']}, {weather['temp']}, {weather['humidity']}, {weather['wind_speed']}, {weather['wind_direction']})"
#         self._cur.execute(sql)
#         self._cur.execute(sql)
#         except Exception as ex:
#             print(f"Error inserting row, ({ex})")
#         return None
#
#     def delete_row(self, row_id, table_name = "weather"):
#         """Delete row into weather db table"""
#         self._cur.execute(f"DELETE FROM {table_name} WHERE id={row_id}")
#         self._cur.execute(sql)
#         except Exception as ex:
#             print(f"Error accessing {table_name} ({ex})")
#         return None
#
#     def delete_all_rows(self, table_name = "weather"):
#         """Delete all rows in the the table"""
#         self._cur.execute(f"DELETE FROM {table_name}")
#         self._cur.execute(sql)
#         except Exception as ex:
#             print(f"Error accessing {table_name} ({ex})")
#         return None
#
#     def commit(self):
#         """Commit changes to database"""
#         self._db_conn.commit()
#         return None
#
#     def close(self):
#         self._db_conn.close()
#         return None
#
#     # Example of operator overloading
#     def __str__(self):
#         """Return status of DB connection"""
#         if self._db_conn:
#             return "DB still connected"
#         else:
#             return "DB not connected"
#
#     def __del__(self):
#         """Close connection automatically when object is deleted"""
#         self._db_conn.close()
#         return None
```





```
#!/bin/python
# Name: movies.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will download the top 250 movies from IMDB
# and will allow the user to display the top-n ranked movies or search for
# their favourite movies.

"""
Download and display the Top 250 Movies Data from IMDB.
"""

import sys
import imdb
import re

movie_menu = """
Movies Menu
-----
1. Get online movie ranking from IMDB
2. Display top ranking movies
3. Search for movie
"""

def get_movies():
    """
    Get the top 250 movies online from IMDB
    """
    ia = imdb.IMDb()
    movies = {}

    for rank, movie in enumerate(ia.get_top250_movies(), start=1):
        movies[rank] = str(movie)

    return movies

def display_movies(movies, top):
    """
    Display top movies in a given dict
    """
    if not movies:
        print("No movies")
    else:
        for rank, name in movies.items():
            print(f"({rank}>4d) {name}<30s")
            if int(rank) == int(top):
                break
    return None

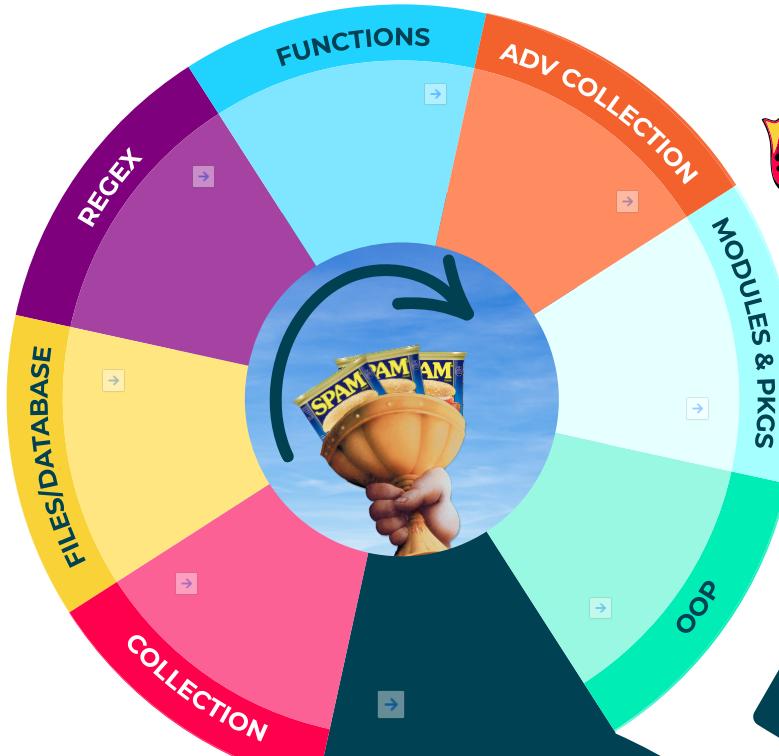
def search_movies(pattern=r".", movies=None):
    """
    Search for pattern in a given dict of movies
    """
    if movies is None:
        movies = {}
    if movies:
        for rank, name in movies.items():
            m = re.search(pattern, name, re.IGNORECASE):
            if m:
                print(f"({rank}>4d) {name}<30s")
    else:
        print("No movies to search")
    return None

def menu():
    """
    Movie Menu
    """
    films = {}
    while True:
        print(movie_menu)
        option = input("Enter option (1-3, [q=quit]): ")
        if option == "1":
            films = get_movies()
        elif option == "2":
            max = input("How many top movies do you want [default=250]: ")
            if not max:
                max = 250
            display_movies(films, max)
        elif option == "3":
            pattern = input("Enter Regex search pattern: ")
            search_movies(pattern, films)
        elif option.lower() == "q":
            break
        else:
            print("Invalid option")
    return None

def main():
    menu()
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```





QUEST

PPLY



# Digital Phase Activities

These activities should have been completed as part of the Digital Phase of your learning

Course Introduction	if, elif and else	len(), min(), max() and sum()
Introduction to Python 3	Indentation and 4 spaces	Mutable and immutable
IDLE & REPL prompt	Comparison operators	Repetition using while and for loops
Variables & objects	Boolean and 'in' operators	Introduction to file handling and 'with'
Comments	Unicode and special chars	Reading and writing text files
PEP 8 Style Guide	str indexing and slicing	Reading and writing csv files
keyboard input()	str concatenation and repetition	Reading and writing to DB files
int, float, str, list and booleans	str methods	Introduction to user functions
type(), dir() and help()	str formatting using f-strings	Introduction to Python Std Library
Casting/conversion	Collections introduction	Importing Modules
Math and string operators	Tuples, lists, dict and sets	Introduction to Unit Testing
Flow control/decision making	Collection indexing and slicing	Introduction to Error Handling

What was your biggest challenge during the Discovery Phase?

Flow controls / tuples dict and sets

Add a Sticky Note

Tuples, lists, dict and sets were tricky for me

Data storage and file handling



# Mini Lab

Refresh digital concepts before main Quest

⌚ 15 mins

During your Digital Phase you completed labs for a simple conversion tool for converting temperatures from Celsius to Fahrenheit. It is common to create simple tools for reading, converting and formatting text data.

In this mini-lab, you will be reading text data from a csv file and displaying the formatted weather information to the screen. You will need to create user functions for converting the temperature from Celsius to Fahrenheit and the wind speed from km/hr to mph.

## Items to consider in your solution:-

1. PEP8 compliance, shebang line and comments.
2. User input for the filename.
3. Appropriate use of flow control.
4. Reading from a text file.
5. Use of error handling where necessary.
6. String formatting using f-strings.
7. Optional use of library module.

```
edinburgh, 548206, 8.3, 868
london, 8961989, 10.8, 690
paris, 11078546, 11.7, 720
rome, 4278350, 15.8, 878
washington, 7766925, 13.7, 1088
berlin, 3567000, 10.1, 669
buenos aires, 15258000, 17.3, 1112
bangkok, 10722815, 27.7, 1207
cape town, 4709990, 16.4, 621
wellington, 381900, 12.5, 926
beijing, 20896820, 12.7, 566
```



City	Pop	Rainfall	Temp
Edinburgh	548206	46.940"	34.17°F
London	8961989	51.440"	27.17°F
Paris	11078546	53.660"	28.35°F
Rome	4278350	60.440"	34.57°F
Washington	7766925	56.660"	42.83°F
Berlin	3567000	50.180"	26.34°F
Buenos Aires	15258000	63.140"	43.78°F
Bangkok	10722815	81.860"	47.52°F
Cape Town	4709990	61.520"	24.45°F
Wellington	381900	54.500"	36.46°F
Beijing	20896820	54.860"	22.28°F

edinburgh, 548206, 8.3, 868

london, 8961989, 10.8, 690

paris, 11078546, 11.7, 720

rome, 4278350, 15.8, 878

washington, 7766925, 13.7, 1088

berlin, 3567000, 10.1, 669

buenos aires, 15258000, 17.3, 1112

bangkok, 10722815, 27.7, 1207

cape town, 4709990, 16.4, 621

wellington, 381900, 12.5, 926

beijing, 20896820, 12.7, 566

City	Pop	Rainfall	Temp
<hr/>			
Edinburgh:	548206	46.940"	34.17°F
London:	8961989	51.440"	27.17°F
Paris:	11078546	53.060"	28.35°F
Rome:	4278350	60.440"	34.57°F
Washington:	7766925	56.660"	42.83°F
Berlin:	3567000	50.180"	26.34°F
Buenos Aires:	15258000	63.140"	43.78°F
Bangkok:	10722815	81.860"	47.52°F
Cape Town:	4709990	61.520"	24.45°F
Wellington:	381900	54.500"	36.46°F
Beijing:	20896820	54.860"	22.28°F
<hr/>			



# Mini Lab Solution/s

Refresh digital concepts before main Quest

```
#!/bin/python
# Name: demo_mini_lab.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will review the topics covered in
# the QA 2.0 Digital session including objects, string handling,
# file interaction, flow control, simple exception handling
# and importing
# Display City weather information.
# ...
def c_to_f ( temp ):
    """ Return temp in Fahrenheit """
    return ( float ( temp ) * 9/5 ) + 32

def mm_to_inch ( height ):
    """ Return precipitation in inches """
    return float ( height ) / 25.4

file = input ("Enter filename: " )

try :
    # Open file handle for Reading in Text mode.
    with open ( file , mode="rt" ) as fh_in :
        # Print header line.
        print ( ("{City": "Pop": "Rainfall": "Temp":") )
        print ( (" " * 43) )

        for line in fh_in :
            city , pop , temp , rain = line . split ( " " ) # Split str into list.
            temp_f = c_to_f ( temp ) # Convert into celcius.
            rain_i = mm_to_inch ( rain ) # Convert into inches.
            print ( ("{" + city + ".title": ( ) >15s) + ("Pop": "10s) + ("Rainfall": "8s) + ("Temp": "8s) + )
            print ( (" " * 43) )

            # Automatically close file handle.
            print ( (" " * 43) )
except :
    print ( ("Error - could not open file" , file ) )
```

```
#!/bin/python
# Name: demo_mini_lab_csv.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will review the topics covered in
# the QA 2.0 Digital session including objects, string handling,
# file interaction, flow control, simple exception handling
# and importing
# ...
# Display City weather information.
# ...
import csv

def c_to_f ( temp ):
    """ Return temp in Fahrenheit """
    return ( float ( temp ) * 9/5 ) + 32

def mm_to_inch ( height ):
    """ Return precipitation in inches """
    return float ( height ) / 25.4

file = input ("Enter filename: " )

try :
    # Open file handle for Reading in Text mode.
    with open ( file , mode="rt" ) as csvfile :
        # Print header line.
        print ( ("{City": "Pop": "Rainfall": "Temp":") )
        print ( (" " * 43) )

        rows = csv . reader ( csvfile , delimiter = " " )
        for line in rows :
            city , pop , temp , rain = line
            temp_f = c_to_f ( temp ) # Convert into celcius.
            rain_i = mm_to_inch ( rain ) # Convert into inches.
            print ( ("{" + city + ".title": ( ) >15s) + ("Pop": "10s) + ("Rainfall": "8s) + ("Temp": "8s) + )
            print ( (" " * 43) )

            # Automatically close file handle.
            print ( (" " * 43) )
except :
    print ( ("Error - could not open file" , file ) )
```

```
#!/bin/python
# Name:      demo_mini_lab.py
# Author:    QA2.0, Donald Cameron
# Revision:  v1.0
# Description: This program will review the topics covered in
# the QA 2.0 Digital session including objects, string handling,
# file interaction, flow control, simple exception handling.
"""
    Display City weather information.
"""

def c_to_f ( temp ):
    """ Return temp in Fahrenheit """
    return ( float ( temp )* 9/ 5)+ 32

def mm_to_inch ( height ):
    """ Return precipitation in inches """
    return float ( height )/ 25.4

file = input ( "Enter filename: " )

try :
    # Open file handle for Reading in Text mode.
    with open ( file , mode="rt" ) as fh_in :
        # Print header line.
        print ( f"{'City' :>15s} { 'Pop' :^10s} { 'Rainfall' :^8s} { 'Temp' :^8s} " )
        print ( "-" * 43 )

        for line in fh_in :
            city , pop , temp , rain )= line . split ( "," ) # Split str into list.
            temp_f = c_to_f ( temp ) # Convert into celsius.
            rain_i = mm_to_inch ( rain ) # Convert into inches.
            print ( f" {city :>15s} : {pop :<10s} "
                    f" {temp_f :<5.3f} {double prime} "
                    f" {rain_i :>5.2f} {degree fahrenheit} " )
        # Automatically close file handle.
        print ( "-" * 43 )
except :
    print ( f"Error - could not open file {file }" )
```

```
#!/bin/python
# Name:    demo_mini_lab_csv.py
# Author:  QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will review the topics covered in
# the QA 2.0 Digital session including objects, string handling,
# file interaction, flow control, simple exception handling
# and importing.
"""
    Display City weather information.
"""
import csv

def c_to_f ( temp ):
    """
    Return temp in Fahrenheit """
    return ( float ( temp ) * 9/5 ) + 32

def mm_to_inch ( height ):
    """
    Return precipitation in inches """
    return float ( height ) / 25.4

file  = input ( "Enter filename: " )

try :
    # Open file handle for Reading in Text mode.
    with open ( file , mode="rt" ) as csvfile :
        # Print header line.
        print ( f"{'City' :>15s} {'Pop' :>10s} {'Rainfall' :>8s} {'Temp' :>8s} " )
        print ( "-" * 43 )

        rows = csv.reader ( csvfile , delimiter =',' )
        for line in rows :
            ( city , pop , temp , rain )= line
            temp_f = c_to_f ( temp ) # Convert into celsius.
            rain_i = mm_to_inch ( rain ) # Convert into inches.
            print ( f" {city .title () :>15s} : {pop :<10s} "
                    f" {temp_f :<5.3f} \N{double prime} "
                    f" {rain_i :>5.2f} \N{degree fahrenheit} " )
            # Automatically close file handle.
        print ( "-" * 43 )
except :
    print ( f"Error - could not open file {file }" )
```



# Questions...

Now is your chance to clear up loose ends from the Digital Phase

Short Question  
& Review

<Add a  
Sticky Note>



# PyCharm

Lets get ready to code



**Download Community Edition from here:**

<https://www.jetbrains.com/pycharm/download/#section=windows>

**Documentation for PyCharm can be found here:**

<https://www.jetbrains.com/help/pycharm/quick-start-guide.html>



PyCharm is a dedicated Python IDE providing more functionality and tools and more engaging experience for developers including project management, virtual environments and GitHub integration.



## 3 Editions

- Community - Free
- Professional (data Science and version control)
- Edu (Learning)

## Multi-platform



## Cool Features

- Python shell command line with colour coded highlighting and error messages
- Multi-tab text editor with colour coded highlighting, auto indentation and completion
- Search and replace tool
- Powerful debugger with breakpoints
- Support virtual environments
- Support Projects and Packages
- Integrated support for GitHub, Mercurial and Subversion etc.



# PyCharm

Lets get ready to code



Mostly everything you do in **PyCharm** revolves around **Projects**.

They act as a basis for grouping related code, allowing refactoring/rename within the group, better management of modules etc.



Start Pycharm CE (Community Edition)

Create a new Project and choose folder location inside **C:\labs**.

Select VirtualEnv to create an isolated install of Python with its own Interpreter and modules.

Choose a Base Interpreter and select Inherit Global Site Packages box.

File>**Settings**

File>Settings>Editor>**Font**

File>Settings>Editor>Font>**Color Scheme**

File>Settings>Editor>Font>File and Code Templates>**Python**

Review Navigation Bar and Editor.

Project>New>**Python File**

Compile and run script.



**Not  
confident**

**Quite  
confident**

**Very  
confident**

Back to  
the quest





# Collections

Storing things in an ordered or unordered way

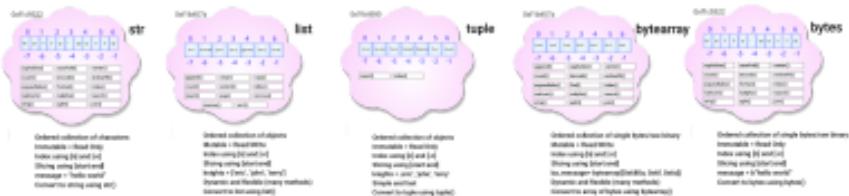
Where do I put all these things?



Tuples LISTS DICTIONARIES SETS

# COLLECTIONS

## Ordered Collections



## Unordered Collections



# TUPLES



```
#!/bin/python
# Name:    demo_tuples.py
# Author:  QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to create and access a tuple
# which is an Immutable (Read-only) Ordered Collection of objects.
```

## Creating, indexing and accessing objects within a Tuple

```
import sys
```

```
def main():
```

```
scottish_bands = ( "The Proclaimers" , "Franz Ferdinand" , "Primal Scream"  
, "Simple Minds" )
```

```
print ( f "Scottish bands is of {type ( scottish_bands ) }" )
print ( f "There are {len ( scottish_bands ) } bands in {scottish_bands }." )
```

print(f"The 1st band is {scottish\_band}")

print {{ "The last band is " }} {{ scottish\_bands }} {{ -1 }} {{ "!" }}

```
print (f "First three bands = { scottish_bands [ 0:3 ] }")
```

```
# Try this without the try/except block.
```

scottish\_bands [2] = "Deacon B

```
    print ("Cannot
```

```
f "at index {scottish_bands . index ('Primal Scream')};"
return None
```

main()

59

# LISTS



```
#!/bin/python
# Name: demo_lists.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to create, grow, shrink and access
# objects within a list.
"""

Playing with a list of numeric (numbers) and non-numeric (movie titles) data
"""

import sys

def main():
    """Create, grow, shrink and play with lists (numeric and non-numeric)"""
    numbers = [ 250, 32, 46, 9, 31, 79, 123 ]
    movies = [ 'Local Hero', 'Gregory's Girl', 'Shallow Grave' ]

    # A few cool built-in functions for finding, length, smallest
    # largest and sum of numbers.
    print ( f"Length of numbers = {len( numbers )}" )
    print ( f"Smallest number = {min( numbers )}" )
    print ( f"Largest number = {max( numbers )}" )
    print ( f"Sum of numbers = {sum( numbers )}" )

    # Some of these functions (Not sum) can also be used on non-numeric data.
    print ( f"Number of movies = {len( movies )}" )
    print ( f"1st movie in char order = {min( movies )}" )
    print ( f"Last movie in char order = {max( movies )}" )

    # We can Index and Slice [start:end] lists.
    print ( f"1st movie = {movies[0]}" )
    print ( f"Last movie = {movies[-1]}" )
    print ( f"Last three movies = {movies[-3:]}" )

    # Let's try some list methods for adding to the list, this
    # and within the list. List will be re-indexed if required.
    movies.append('Trainspotting') # Append one object to RHS.
    movies.extend([ 'Brave', 'Braveheart' ]) # Extend multiple objects to RHS.
    movies.insert( 0, 'Comfort & Joy' ) # Insert one object to LHS (before index 0).
    movies.insert( 3, 'The Angel's Share' ) # Insert on object before index 3.
    print ( f"Scottish movies = {movies}" )

    # Let's try some list methods for removing from the list, this
    # and within the list. List will be re-indexed if required.
    last = movies.pop() # Last object removed and optionally returned.
    first = movies.pop(0) # First object removed and optionally returned.
    movies.pop(3) # Remove object at index 3.
    print ( f"Scottish movies = {movies}" )

    # movies.clear() # Empties List.
    # films = movies.copy() # Copies List.
    # Notice how print parameters can span across lines.
    print ( f"I spotted Trainspotting {movies.count('Trainspotting')}" )
    print ( f"at index {movies.index('Trainspotting')}" )

    # The built-in sorted function returns a sorted list.
    print ( f"Movies sorted = {sorted( movies, key=str, reverse=False )}" )
    print ( f"Movies sorted = {sorted( movies, key=str, reverse=True )}" )
    print ( f"Movies sorted = {sorted( movies, key=len )}" ) # Bespoke sort.

    movies.sort() # Sorts list in place.
    movies.reverse() # Reverses list in place.
    movies.remove('Trainspotting')

    print ( f"List of movies = {movies}" ) # Saturday night viewing.
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# DICTIONARIES



Since 2016, Python 3.6 stores dictionaries as ordered - in the order that the key: objects are assigned, but this may change in a future implementation, so don't assume!

```
#!/bin/python
# Name: demo_dictionaries.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to create, grow and shrink
# dictionaries and use keys and iterator for loops to access objects within.
# Dictionaries are mutable collections that can store multiple objects and
# access using UNIQUE keys.
"""
Weather Program to store and display average (temp, precipitation and humidity)
for UK cities in 2020.
"""
import sys
import pprint

def main():
    """Display weather info for uk cities"""

    # Create a dictionary to store annual average weather information
    # (Mean Temp, Precipitation, Humidity) for UK Cities in 2020.
    weather = { 'Glasgow' : [ 9, 53.6, 87],
                'London' : [ 12, 49.7, 81],
                'Edinburgh' : [ 9, 37.9, 81],
                'Manchester' : [ 7, 35.8, 86],
                'Thurso' : [ 6, 89.6, 85]
    }

    # Add two new key: objects to weather dictionary.
    weather ['Southampton'] = [ 8, 28.4, 93]
    weather ['Inverness'] = [ 6, 23.5, 83]

    # weather.clear() # Empty dictionary.
    # w_info = weather.copy() # Copy dictionary.
    # weather.pop('Thurso', False) # Remove Thurso as not a City.
    # weather.popitem() # Remove next key:object found.

    print ( f"Number of cities recorded = {len ( weather ) }" )
    print ( f"Weather info for Glasgow = {weather [ 'Glasgow' ] }" )
    print ( f"Average precipitation = {weather [ 'Glasgow' ][ 1 ] } mm" )

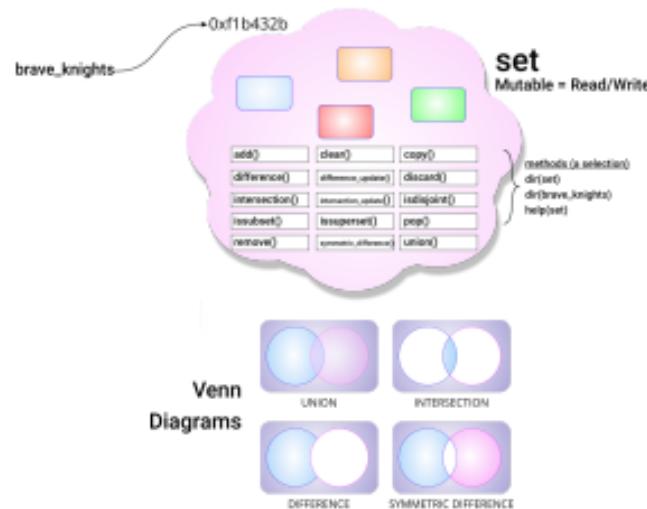
    # ITERATE through the key and objects within the dict.
    # keys() returns a VIEW into next key.
    for city in weather.keys():
        print ( f" {city} average = {weather [ city ] }" )

    # ITERATE through the values within the dict.
    # values() returns a VIEW into the next value.
    for w_info in weather.values():
        print ( f" {w_info }" )

    # ITERATE through the (values, objects) within the dict.
    # items() returns a VIEW into the next pair of (key, value).
    for city, w_info in weather.items():
        print ( f" {city} weather info = {w_info }" )
    return None

if __name__ == "__main__":
    main()
    sys.exit ( 0 )
```

# SETS



There are many Brave Knights who would also like to be Lumberjacks and vice-versa, so we will attempt to combine sets of Knights and sets of Lumberjacks using common SET operators and methods.

```
#!/bin/python
# Name: demo_dictionaries.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to create, grow and shrink
# and combine sets using SET operators and methods.
# Sets are mutable collections that store multiple UNIQUE objects (with no keys).
*** The joy of SETS!
*** import sys
def main():
    """ Combining collections of Brave Knights and Lumber Jacks """
    # Create two sets, one with objects and one empty.
    brave_knights = { 'donald' , 'brian' , 'terry' , 'eric' , 'michael' }
    lumber_jacks = set () # Creates an empty set.

    # Sets are Mutable so can be grown (add) and shrunk(pop/remove).
    brave_knights . add ('john' )
    brave_knights . add ('terry' ) # Sets automatically remove duplicates!
    lumber_jacks . add ('donald' )
    lumber_jacks . add ('michael' )
    lumber_jacks . add ('graham' )

    # brave_knights.pop() # Remove next object found.
    # brave_knights.remove(") # Remove object by name.

    # Sets can be combined using operators or methods. Remember Venn diagrams?
    # Combine SETS using SET methods.
    print ( f "Brave Knights and Lumber Jacks = {brave_knights . union (lumber_jacks ) }" )
    print ( f "Brave Lumber Knights = {brave_knights . intersection (lumber_jacks ) }" )
    print ( f "Strictly Brave Knights ONLY = {brave_knights . difference (lumber_jacks ) }" )
    print ( f "Everyone BUT Brave Lumber Knights = {brave_knights . symmetric_difference (lumber_jacks ) }" )
    print ( f " " * 60 )
    # Combine SETS using SET operators.
    print ( f "Brave Knights and Lumber Jacks = {brave_knights | lumber_jacks }" )
    print ( f "Brave Lumber Knights = {brave_knights & lumber_jacks }" )
    print ( f "Strictly Brave Knights ONLY = {brave_knights - lumber_jacks }" )
    print ( f "Everyone BUT Brave Lumber Knights = {brave_knights ^ lumber_jacks }" )
    return None

if __name__ == "__main__":
    main()
    sys . exit ( 0 )
```



# Quiz

## Collections Knowledge Check



5-10 mins

### 1. Name three ordered and two unordered collections?

Ordered?

string  
list  
tuple

Unordered

sets  
dictionary

### 2. Is a tuple mutable?

No

### 3. The joy of sets! What would be printed from the following code?

```
>>> scottish_movies = {"Gregory's Girl", "Local Hero", "Comfort and Joy", "That Sinking Feeling"}  
>>> fav_movies = {"The Hobbit", "Local Hero", "The Lord of the Rings"}  
>>> print(scottish_movies & fav_movies)
```

{local hero}

To be revealed

### 4. Fill in the blanks:

Dictionaries have unique

keys

Sets have unique

elements

To be revealed

To be revealed

### 5. What would be printed from the following code?

```
>>> movies = "Gregory's Girl", "Local Hero", "Comfort and Joy", "That Sinking Feeling"  
>>> x, *y, z = movies  
>>> print(y)
```

[local hero, comfort and joy]

To

### 6. Write the print statement from question 3 using set methods?

print(scottish\_movies.intersection(fav\_movies))

To be revealed

# Exercises

⌚ 30 mins

## Collections - exercises for 'brave' Knights

1 Write a Python script called `C:\labs\lotto.py` that will generate and display 6 unique random lottery numbers between 1 and 50. Think about which Python data structure is best suited to store the numbers! Use the Python help() function to find out which function to use from the python standard library called random.

2 Create a new script in `C:\labs\` called `topTen.py`

- Read the top 250 movies from the `C:\labs\top250_movies.txt` file and store them in a list called `movies`.
- Print out the top 10 with rankings (right justified 5 characters) followed by a colon and space followed by name of the film.
- Print out the first and last movie.
- Modify the code to allow the user to choose the top `N` to be displayed.

## Collections - stretch exercises for Kings

3 Create a reusable user function called `get_movies` which returns all the movie data into a list called `movies`.

4 Create another function called `display_movies` which accepts two parameters - a list of movies, and number of movies to be displayed.

5 Create another function called `search_movies` which accepts a simple string (to search), and the list of movies to search.

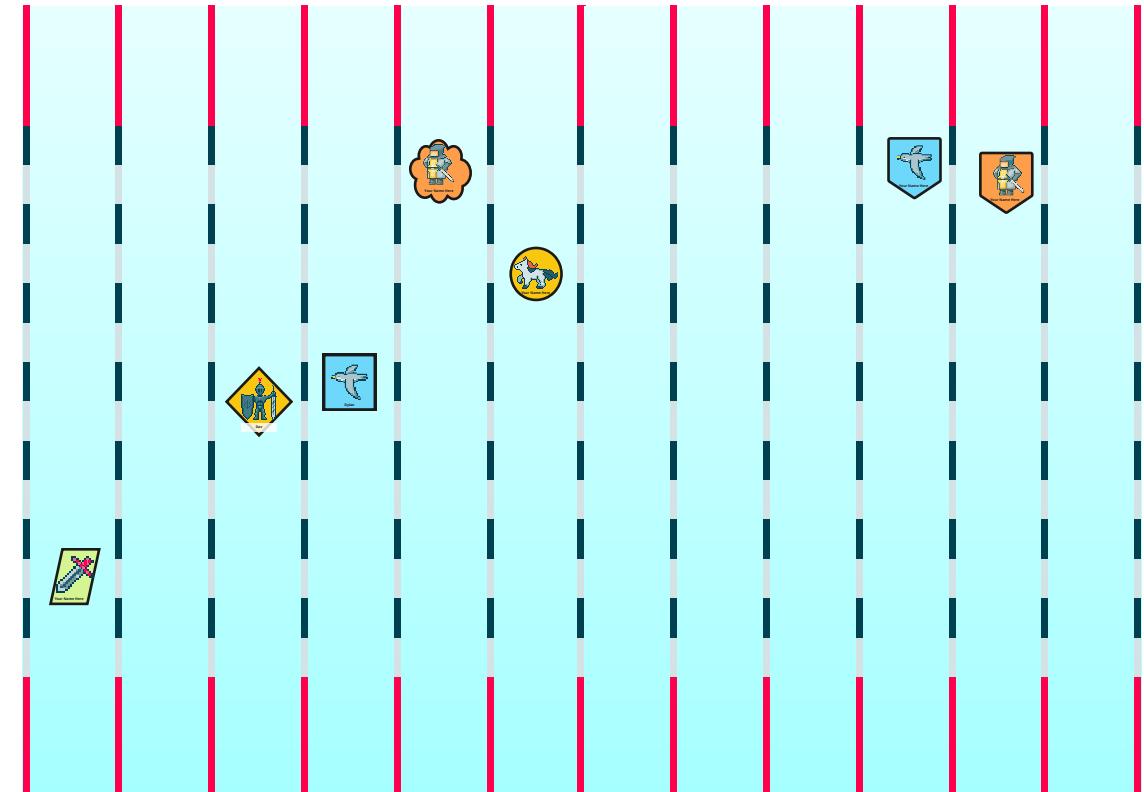
6 Now create a simple user menu which prompts the user for the following:

### Movies Menu

- 
1. Get online movie ranking from IMDB
  2. Display top ranking movies
  3. Search for movie
- Enter option (1-3, [qQ=quit]):

7 Extra kudos for having a main function and using the `__name__ == "__main__"` namespace trick.

Move your piece down as you complete the exercises



**Not  
confident**

**Quite  
confident**

**Very  
confident**

[Back to  
the quest](#)





# Persistent Data

# File Interaction

# Sequential Access

```
#!/bin/python
# Name:    demo_sequential_rw.py
# Author:  QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to Interact with the file
# system by opening/closing file handles for read, write and appending.
"""
    Write and read movie data to a text file.
"""

import sys

def main():
    """ Demonstrate writing and reading to a text file. """
    movies = { 'Donald' : [ 'Braveheart' , 'Brave' , 'Brigadoon' ] ,
               'Mira'  : [ 'Matrix' , 'Mad Max' , 'Magnolia' ] ,
               'Sarah' : [ 'Seven' , 'Scream' , 'Saving Private Ryan' ] }
    filename = r"C:\abs\movies.txt"      # Always use a raw string for paths.

    # Open file handle for WRITING in text mode.
    fh_out = open( filename , mode="wt" )

    # Iterate through Movie keys and write key: objects to file handle.
    for name in movies.keys():
        print( f"{name}: {movies[name]}", end="\n" )
        fh_out.write( f"{name}: {movies[name]}\n" ) # Remember newline.
    fh_out.close() # Flush buffers and close file handle.

    # Open file handle for READING in text mode.
    fh_in = open( filename , mode="rt" )

    # text = fh_in.read() # Read ENTIRE file into str object.
    # text = fh_in.read(30) # Read NEXT 30 chars into str object.
    # text = fh_in.readline() # Read NEXT line into str object.
    # print(text)

    # lines = fh_in.readlines() # Read ENTIRE file into list object.
    # print(lines)
    # print(f"1st line = {lines[0]}")

    # Iterate through file handle and read one line at a time.
    # for name in open(filename, mode="wt"):
    for name in fh_in:
        print( name, end="" )
    fh_in.close() # Flush buffers and close file handle.
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# File Interaction

## TEXT Random Access

```
#! /bin/python
# Name:    demo_random_rw_text.py
# Author:  QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to Interact with the
# file system by opening/closing file handles random reading using the
# seek() and tell() file handle methods.

"""
    Read movie data randomly from a TEXT file.
"""

import sys

def main():
    """ Demonstrate random reading from a TEXT file. """

    filename = r"C:\I\abs\movies.txt"      # Always use a raw string for paths.
    SOI = 0 # Start of file.
    CUI = 1 # Current Position (only on binary file).
    EOI = 2 # End of file (only on binary file).

    # Open file handle for READING in TEXT mode.
    fh_in = open( filename , mode="rt" )

    # Seek to char position 50 from SOF and read next 30 chars.
    fh_in . seek ( 50, SOI)
    text = fh_in . read ( 30)
    print ( f"Text = {text} at position {fh_in . tell () - len ( text )}" )

    # Seek to char position 90 from SOF and read next 40 chars.
    fh_in . seek ( 90, SOI)
    text = fh_in . read ( 40)
    print ( f"Text = {text} at position {fh_in . tell () - len ( text )}" )

    fh_in . close () # Flush buffers and close file handle.

    return None

if __name__ == "__main__":
    main()
    sys . exit ( 0)
```

# File Interaction

## BINARY Random Access

```
#!/bin/python
# Name:    demo_random_rw_binary.py
# Author:  QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to Interact with the
# file system by opening/closing file handles random reading using the
# seek() and tell() file handle methods.

"""
    Read movie data randomly from a binary file.
"""

import sys

def main():
    """
        Demonstrate random reading from a BINARY file.
    """

    filename = r"C:\abs\movies.txt"      # Always use a raw string for paths.
    SOF = 0 # Start of file.
    CUI = 1 # Current Position.
    EOF = 2 # End of file.

    # Open file handle for READING in BINARY mode.
    fh_in = open( filename , mode="rb" )

    # Seek to char position 50 from SOF and read next 30 bytes.
    fh_in . seek ( 50, SOF)
    text = fh_in . read ( 30)
    print ( f"Text = {text} at position {fh_in . tell () - len ( text )} ")

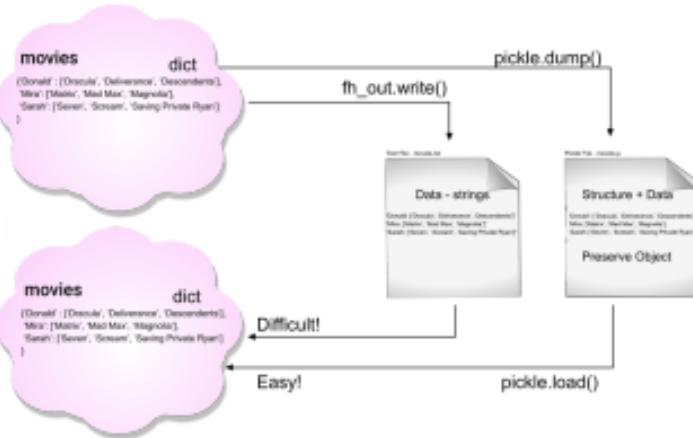
    # Seek forwards 30 chars from current position and read next 40 bytes.
    fh_in . seek ( 30, CUI)
    text = fh_in . read ( 40)
    print ( f"Text = {text} at position {fh_in . tell () - len ( text )} ")

    # Seek backwards 40 chars from EOF and read next 20 bytes.
    fh_in . seek ( - 40, EOF)
    text = fh_in . read ( 20)
    print ( f"Text = {text} at position {fh_in . tell () - len ( text )} ")

    fh_in . close () # Flush buffers and close file handle.
    return None

if __name__ == "__main__":
    main()
    sys . exit ( 0)
```

# Pickle an Object



Pickling converts any Python object, except functions, into a stream of bytes - preserving the Data plus its **STRUCTURE**!

Resultant pickle files can be written in a number of formats - this is called the pickle protocol and is a named parameter for the pickle dump function. Choose from:

- protocol=0 (ASCII for backwards compatibility)
- protocol=1 (Binary)
- protocol=2 (Binary from Python 2.3)
- protocol=3 or pickle.DEFAULT\_PROTOCOL (Binary from Python 3.0)
- protocol=4 or pickle.HIGHEST\_PROTOCOL (Binary from Python 3.4)

```
#!/bin/python
# Name: demo_pickle.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to preserve Python objects
# (data plus structure) to a file using the pickle module.

# Preserve/Pickle ONE Python object to a file.

import sys
import pickle
import pprint

def main():
    """ Preserve and reload a Python object to a file """
    # Create a dict to hold our favourite movies.
    movies = { 'Donald' : [ 'Dracula' , 'Deliverance' , 'Descendants' ] ,
               'Mira' : [ 'Matrix' , 'Mad Max' , 'Magnolia' ] ,
               'Sarah' : [ 'Seven' , 'Scream' , 'Saving Private Ryan' ] }

    # Open file handle in binary write mode for pickling.
    fh_out = open ( r "C:\1 abs\movies.p" , "wb" )

    # Dump the movies dict to the pickle file using a chosen protocol.
    # pickle.dump(movies, fh_out, protocol=0) # ASCII
    # pickle.dump(movies, fh_out, protocol=4) # binary space efficient
    pickle . dump(movies, fh_out, pickle . DEFAULT_PROTOCOL) # protocol=3
    # pickle.dump(movies, fh_out, pickle.HIGHEST_PROTOCOL) # protocol=4
    fh_out . close ()

    # Open file handle in binary read mode for pickling.
    fh_in = open ( r "C:\1 abs\movies.p" , "rb" )

    # Reload pickled dict from file back into memory.
    films = pickle . load ( fh_in )
    fh_in . close ()

    # Display and compare the movies and films dict using the
    # Std Py library module pprint (pretty print)
    pprint . pprint ( movies )
    print ( "-" * 60 )
    pprint . pprint ( films )
    return None

if __name__ == "__main__":
    main()
    sys . exit ( 0 )
```

# Multiple objects on a Shelf

```
#!/bin/python
# Name: demo_shelfe.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to preserve MULTIPLE
# Python objects to a file using the shelf module.
# Preserve ONE Python object to a file.
# Import modules
import sys
import shelf
import pprint

def main():
    """ Preserve and reload multiple Python objects to a file """
    # Create dictionaries to hold our favourite movies, tv series and music bands.
    movies = { 'Donald' : [ 'Dracula' , 'Deliverance' , 'Descendants' ] ,
               'Mira' : [ 'Matrix' , 'Mad Max' , 'Magnolia' ] ,
               'Sarah' : [ 'Seven' , 'Scream' , 'Saving Private Ryan' ] }
    tv = { 'Donald' : [ "Dad's Army" , "Dragon's Den" , 'Dallas' ] ,
           'Mira' : [ 'MASH' , 'MasterChef' , 'Mythbusters' ] ,
           'Sarah' : [ 'Scrubs' , 'Sesame Street' , 'Star Trek' ] }
    music = { 'Donald' : [ 'David Bowie' , 'Deacon Blue' , 'Duran Duran' ] ,
              'Mira' : [ 'Metallica' , 'Madness' , 'Meatloaf' ] ,
              'Sarah' : [ 'S Club 7' , 'Spandau Ballet' , 'Spice Girls' ] }

    # Open file handle in preserving multiple objects.
    db = shelf . open ( r "C:\lab\movies.db" )

    # Write key: objects to shelf database.
    db[ 'movies' ] = movies
    db[ 'tv' ] = tv
    db[ 'music' ] = music

    db. close ( )

    # Open the Shelf file handle again.
    db = shelf . open ( r "C:\lab\movies.db" )

    while True :
        # Iterate through db and display itemised keys.
        for key in db:
            print ( key )

        # Prompt user to choose key.
        db_key = input ( "Choose a key to display or q=quit: " )
        if db_key . lower ( ) == "q": break
        pprint . pprint ( db[ db_key ] )

    db. close ( )
    return None

if __name__ == "__main__":
    main()
    sys . exit ( 0 )
```

# Interacting with a DB

```
#!/bin/python
# Name: demo_sqlite.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to connect to a SQLite database
# file, create a table, insert and query rows using the SQLite3 module/API.
"""
Create and populate a SQLite database file.
"""
import sys
import sqlite3
import os

def main():
    """ Create, insert and query all rows in database """
    # Prompt user if they want to create or use existing database file
    if os.path.isfile(r"C:\abs\movies_db.sqlite"):
        response = input("Do you want to use existing SQLite file (y/n)? ")
        if response.lower() == "n":
            os.remove(r"C:\abs\movies_db.sqlite")

    # Connect into SQLite database file.
    db_conn = sqlite3.connect(r"C:\abs\movies_db.sqlite")
    # Open a cursor to store
    cur = db_conn.cursor()

    # Create a movies table if it doesn't already exist
    cur.execute("""CREATE TABLE IF NOT EXISTS movies
        (id INTEGER PRIMARY KEY
        ,name TEXT
        ,year DATE
        ,rating INTEGER)""")

    # Loop and write users favourite movies to database file.
    while True:
        movie = input("Enter name of movie (q=quit): ")
        if movie.lower() == "q": break
        year = input("Enter year of release for " + movie + ": ")
        rating = input("Enter your rating for " + movie + " (1-10): ")

        cur.execute("""INSERT INTO movies (name,year,rating) VALUES(?, ?, ?)""", (movie, year, rating))
        print("1 row inserted")

    db_conn.commit() # Commit Changes to database

    # Query, fetch and display all rows from database file.
    cur.execute("""SELECT * FROM movies""")
    rows = cur.fetchall()
    for row in rows:
        print(row)

    db_conn.close() # Close database connection.
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```



# Quiz

Persistent Data Knowledge Check



5-10 mins

1. What is the default mode for the built-in open() function?

"rt" = Read and Text mode

2. What is the default buffering mode for text files?

Line Buffering

3. Which methods are used for random access?

seek, read and tell

Can use mmap to map file into memory

4. Which Python standard library module preserves **ONE** Python object to a file?

Pickling

5. Which Python standard library module preserves **MULTIPLE** Python objects to a file?

Shelving

To be revealed

# Exercises

⌚ 35 mins

## Persistent Storage - exercises for 'peasants'

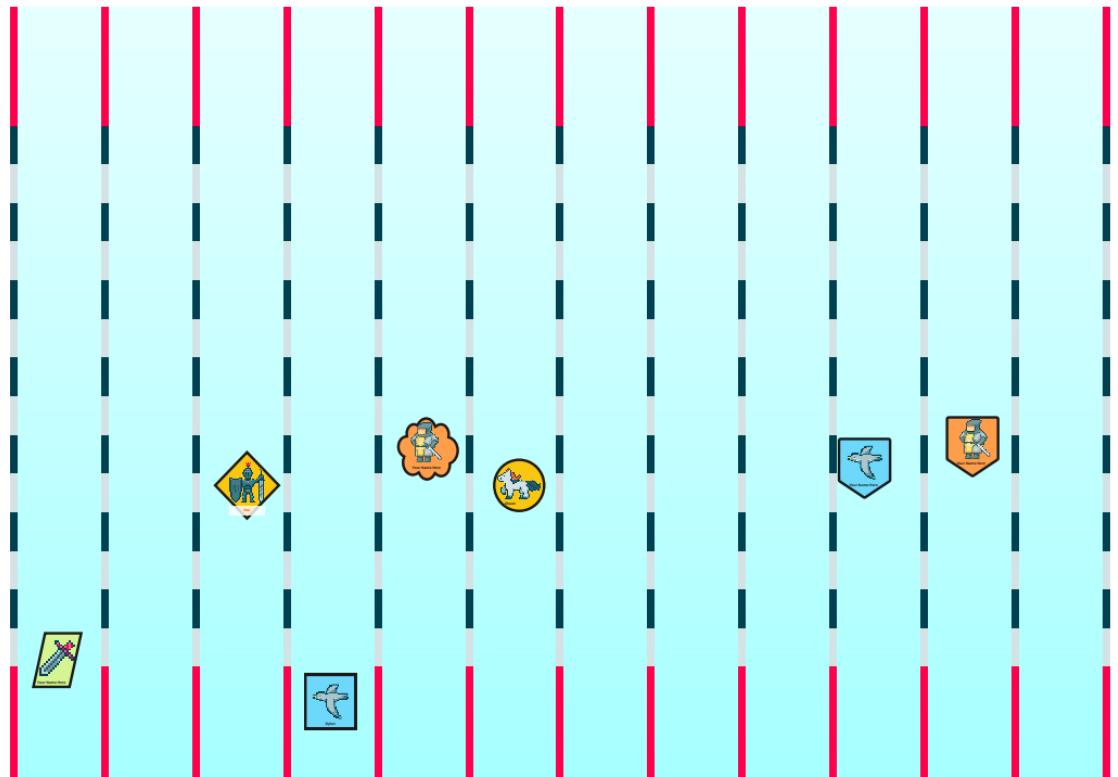
- 1 Open the file `C:\labs\top250_movies.txt`. Notice it contains the top 250 movies as voted by IMDb online users.
- 2 Create a script called `display_movies.py` in the `C:\labs\` folder that will **open** and retrieve the movies one at a time. Display the movies names and remember to **close** the file when finished. Use f-strings to display the movie information.
- 3 You may have noticed that the movie titles are in lower case. Modify the script so that each word of the movie title is capitalised.
- 4 Modify the script to display the ranking from 1 to 250 next to the movie title. Use the built-in `enumerate()` function to generate the ranking.

## Persistent Storage - stretch exercises for Kings

- 5 Create a new script `C:\labs\movie_data.py`. Using the top 5 movies from the previous exercise, create a LIST called `movies` which has five dictionaries with the following keys: "title", "director", and "year" released. Movie information can be found at <http://www.imdb.com>. Write a loop to iterate through the list of dictionaries. Write out the formatted movie information to a file called `C:\labs\top5_movie_info.txt`. Use the print function to write instead of the `write` method.  
For example:  
`1994 - Title: The Shawshank Redemption Director: Frank Darabont  
1972 - Title: The Godfather Director: Francis Ford Coppola`
- 6 Think how difficult it would be to reload this information back into a suitable Python data structure. Copy the `C:\labs\movie_data.py` file to `C:\labs\movie_dict.py` and modify the new script to **preserve** the movie list to a compressed pickle file called `C:\labs\top5movies.pgg`. Load the object back into memory and compare with the original movies dictionary [Hint: use `pprint`].

- 7 And now for something a little different. Write a Python script called `C:\lab\display_unused_ports.py` that lists all the unused port numbers between 1 to 200 in the SERVICES file. This file is used by applications and TCP/IP to translate network service names into port numbers, for example, http and port 80. Your script should take into account which platform it is running on and select `/etc/services` on Linux and `C:\WINDOWS\System32\drivers\etc\services` on Windows.  
Some hints to get you started:
  - Test platform and use correct file
  - Write code to iterate through file and display one line at a time
  - ignore comments
  - We've not met Regular Expressions yet, so consider using string splitting to isolate the number
  - Extract port numbers (will have to convert to integer)
  - Port numbers can be duplicated so store numbers in a suitable object which filters duplicates

Move your piece down as you complete the exercises



Not  
confident

Quite  
confident

Very  
confident

Back to  
the quest





# Regular Expressions

match groups  
assertions pre-compile  
group basic end substitution  
repetition  
escape expressions  
groupings regex regular extended  
alternation start  
objects quantifiers

Have you used Regular  
Expressions before?  
Comment where you  
have used them.

*<Add a  
Sticky Note>*

# Basic Regex

**Regular Expressions** (Regex, or just RE) are used in many tools, sql and programming languages, and have their roots in Unix command line tools such as ed, ex, grep, sed, and awk. They provide a more extensive and powerful set of meta-characters for pattern matching and come in several dialects. The Python standard library module `re.py` provides matching functions and support for **Basic**, **Extended** and **Python** pattern characters.

To learn how to use Regex, we need to do two things - learn the theory behind the meta-characters, and then how to apply them using Python code.

The website <https://regex101.com/> is useful for learning and testing your Python Regular Expressions.

## Basic Regular Expressions B.R.E

### Line Anchors

Start `^The`  
End `ing$`

### Grouping/Back Referrals

c i v i c  
r o t o r  
m a d a m  
groupgroup  
^ (.) (.) . \2 \1 \$

### Single Char Class

Any char `^.ing$`  
Any char `^. . . . $`

T E L N E T  
group  
^ ([A-Z]).\* \1 \$

### Single Char Range Class

Char set `^ [rdz] ing $`  
Char range `^ [A-Z] ing $`  
Char ranges `^ [A-Za-z] ing $`  
Char NOT set `^ [^dz] ing $`  
Multiple sets `[aeiou][aeiou][aeiou]`

### Escape Char Class

Escape char `\.`

### Repetition Class

0 or more `:[0-9]*:`  
A diagram below shows the matches for the string '42':  
- The first character '4' is matched by the first character of the first group.  
- The second character '2' is matched by the second character of the first group.  
- The final character ':' is matched by the final character of the first group.

### Limiting Repetition

Exact `:[0-9]{10}:`  
Min, max `:[0-9]{10,20}:`  
At least `:[0-9]{10,}:`

# Extended RE

Extended Regular Expressions or E.R.E added some extra repetition chars (?+), groupings (rhubarb)+ and alternation (or).

## Extended Regular Expressions E.R.E

### Repetition Class

0 or more	" : [0-9] * : "
0 or once	" : [0-9] ? : "
1 or more	" : [0-9] + : "

### Alternation Class

or	" eric graham michael "
----	-------------------------

### Grouping

```
" a (bottle|glass|keg) of (lager|wine|beer)"  
" a bottle of lager"  
" a glass of wine"  
" a glass of beer"
```

## Python Assertions/Escape chars

### Escape Chars

<b>Digit</b>	\d	[0-9]	\D	[^0-9]
<b>Word char</b>	\w	[A-Za-z0-9]	\W	[^A-Za-z0-9]
<b>Whitespace</b>	\s	[ \t\r\f\n]	\S	[^ \t\r\f\n]
<b>Word boundary</b>	\b	[ \t\r\f\n-+()!?.]	\B	[^ \t\r\f\n-+()!?.]
<b>Line anchors</b>	\A	^	\Z	\$

# Regex Code

There are many online resources for learning Regular Expressions with Python. Here are a few to start with:

<https://regex101.com/>

<https://pythex.org/>

<https://extendsclass.com/regex-tester.html>

<http://www.pyregex.com/>

```
#!/bin/python
# Name:      demo_regex_1.py
# Author:    QA2.0, Donald Cameron
# Revision:  v1.0
# Description: This program will demonstrate how to match string
# data using Regular Expressions.

"""
Search a given file using regular expressions and display
matched lines.
"""

import  sys
import  re

def  main():
    """ Demonstrate different 4_Regular_Expressions patterns """
    fh_in  =  open ( r "C:\\" + abs \words" ,  mode="rt" )

    # Iterate through file handle
    for  line  in  fh_in :
        # m = re.search(r"\bthe\b", line) # Match lines starting with 'the'.
        # m = re.search(r"\bing\b", line) # Match lines ending with 'ing'.
        # m = re.search(r"\b[A-Z]\b", line) # Match lines starting with a capital.
        # m = re.search(r"\.\b", line) # Match lines with a dot.
        # m = re.search(r"\b[aeiou][aeiou][aeiou]\b", line) # Match 3 consecutive vowels.
        # m = re.search(r"\b[aeiou]\{5,\}\b", line) # Match at least 5 consecutive vowels.
        # m = re.search(r"\b.....\b", line) # Match 19 char lines.
        # m = re.search(r"\b.{19}\$\b", line) # Match 19 char lines.
        # m = re.search(r"\b(.)(.).2\$1\$2\b", line) # Match lines 5 char palindromes.
        # m = re.match(r"\b(.)(.).2\$1\$2\b", line) # Match 5 char palindromes using match().
        n =  re . search ( r "\b(. )*\1\$" ,  line ) # Match lines starting/ending with same char.

        if  n:
            print ( line ,  end="" )

    fh_in . close ()
    return  None

if  __name__  ==  "__main__":
    main()
    sys . exit ( 0 )
```

# Regex Groups

The functions `search()`, `match()`, `fullmatch()`, `sub()`, and `subn()` either return `None` or an `'re.Match'` object if a pattern is matched. The `match` object has several methods for returning information about the pattern match.

- `m.group()` # Returns the matched pattern.
- `m.start()` # Returns the char position where pattern starts.
- `m.end()` # Returns the char position where the pattern ends.
- `m.groups()` # Returns a tuple of groupings, or error if no groupings in pattern.
- `m.groups()[0]` # Returns the 1st grouping from the tuple. Index = 0, grouping = 1st. Index = 1, grouping = 2nd.
- `m.group(1)` # Also returns the 1st grouping and is more Pythonic. That is, parameter = 1, grouping = 1st.

```
#!/bin/python
# Name: demo_regex_2.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to match string
# data using Regular Expressions
"""
Search a given file using regular expressions and display
matched lines and groupings.
"""

import sys
import re

def main():
    """ Demonstrate 4_Regular_Expressions groupings """
    fh_in = open(r"C:\V\abs\words", mode="rt")

    # Iterate through file handle.
    for line in fh_in:
        # Match 5 char palindromes.
        n = re.search(r"^(.).(\.).\2\1$", line) # Return None or RE Match object.
        if n:
            # The match object m has several methods with info about the match.
            print(f"Matched {n.group()} at char pos {n.start()} - {n.end()}, "
                  f"with groups {n.groups()}, "
                  f"first group is {n.groups()[0]}, " # Index tuple 0 = 1st
                  f"or {n.group(1)}") # More Pythonic way as 1 = 1st.

    fh_in.close()
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# Pre-compile

Notice that in the previous demo we are compiling the pattern for every loop and line that is read from the file. So, if the file has 42k lines then the pattern will be compiled (into low level byte code) 42,000 times - even though the pattern never changes for the life of the loop and file handle.

In this case, there would be significant performance improvements in pre-compiling the pattern just once before the loop. The `re.py` module has a function called `compile()` which returns a Regex Object with the compiled pattern stored within. This object inherits the `search()`, `match()`, and `fullmatch()` functions.

```
#!/bin/python
# Name:    demo_regex_3.py
# Author:  QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to match string
# data using Regular Expressions and pre-compiling the pattern once.
"""
Search a given file using regular expressions and display
matched lines.

"""

import sys
import re

def main():
    """ Display lines in a given file which match Regex """
    fh_in = open(r"C:\abs\words" , mode="rt" )

    # Pre-compile the pattern once if it does not change.
    reobj = re.compile(r"^\s{19}\$")

    # Iterate through file handle
    for line in fh_in :
        # Pre-compiled pattern stored in reobj object.
        n = reobj .match( line ) # Return None or RE Match object.
        if n:
            print( line , end="" )

    fh_in .close()
    return None

if __name__ == "__main__":
    main()
    sys .exit( 0)
```

# Substitution

The `re.py` module also provides two functions for performing substitutions within a string. The `sub()` function can search, make a change, and return a modified string. Whilst the `subn()` function returns a tuple of the modified string and the number of changes made. All the RE functions support optional flags.

## Optional Flags

Long Name	Short Name	Embedded	Comment
<code>re.IGNORECASE</code>	<code>re.I</code>	<code>(?i)</code>	Case insensitive match
<code>re.MULTILINE</code>	<code>re.M</code>	<code>(?m)</code>	<code>^</code> and <code>\$</code> match start and end of line
<code>re.DOTALL</code>	<code>re.S</code>	<code>(?s)</code>	<code>Dot '.' also matches a newline '\n'</code>
<code>re.VERBOSE</code>	<code>re.X</code>	<code>(?x)</code>	Whitespace is ignored within pattern - to allow embedded comments.

```
#!/bin/python
# Name: demo_regex_4.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to match and substitute string
# data using the re.py module and the sub() and subn() functions. It will also
# introduce Regex flags such as ignore case.
# Search and substitute strings using regular expressions.
import sys
import re

def main():
    """ Executed directly when run as a program """
    # This str object is storing login data for the root user from a
    # a file /etc/passwd on the Linux file system.
    line = "root:x:0:0:The root User/root:/bin/sh"

    # The sub() function returns the modified string.
    line = re.sub(r"\[rR\]oot \[Uu]ser", r"Administrator", line)
    print(f"Modified line = {line}\n")

    # The sub() function returns the modified string.
    line = re.sub(r"/bin/sh", r"/bin/bash", line)
    print(f"Modified line = {line}\n")

    # 4. Regular Expressions are GREEDY and match the largest pattern by default.
    line = "root:x:0:0:The root User/root:/bin/sh"
    line = re.sub(r"^( root )\.*", r"I am greedy", line)
    print(f"Modified line = {line}\n")

    # The subn() function returns a TUPLE of (modified string, num changes).
    line = "root:x:0:0:The root User/root:/bin/sh"
    (line, num) = re.subn(r"\[rR\]oot", r"Groot", line)
    print(f"Modified line = {line} with {num} changes\n")

    # Search and replace string with optional RE flags (introduced Py3.1).
    line = "root:x:0:0:The root User/root:/bin/sh"
    # The optional flags can be set several different ways.
    line = re.sub(r"root user", r"Administrator", line,
                  flags=re.IGNORECASE|re.MULTILINE)
    line = re.sub(r"(?im)root user", r"Administrator", line,
                  flags=re.IGNORECASE|re.MULTILINE)
    line = re.sub(r"(?i:r)oot (?iu)ser", r"Administrator", line) # Python 3.6.
    print(f"Modified line = {line}\n")

    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```



# Quiz

## Regular Expression Knowledge Check



### 1. What does B.R.E and E.R.E stand for?

Basic Regular Expression & Extended Regular Expression

### 2. What prefix should you use with your Regex patterns?

the letter "r" (raw text). Its use nullifies escape characters.

### 3. Can you think of a better way to write the following pattern, which is 15 chars long?

`r"^.{15}$"`

`r"^.{15}$"`

### 4. Given this code, what will the pattern match?

```
>>> text = "Brave Sir Robin ran away. Bravely ran away away. When danger
reared it's ugly head, he bravely turned his tail and fled."
>>> m = re.search(r"(brave).*\1", text, flags=re.IGNORECASE)
>>> print(m.group())
```

Brave Sir Robin ran away. Bravely ran away away. When  
danger reared it's ugly head, he brave

\* is the greedy quantifier (match as much as possible)  
\*? is the lazy quantifier

### 5. Given this code, what would be printed?

```
>>> text = "racecar"
>>> m = re.search(r"^(.).(.).\3\2\1$", text)
>>> print(m.groups(), m.groups()[0], m.group(1))
```

('r', 'a', 'c') r r  
regex compares first 3 chars with last 3 chars

# Exercises

⌚ 40 mins

## Regular Expressions - exercises for 'brave' knights

1

For the following exercises, we will be installing the IMDbPY Python package (<https://pypi.org/project/IMDbPY/>). This package is used to retrieve and manage data of the IMDb online movie database about movies, people, characters and companies.

Install the IMDb Package, on command line or within Pycharm.  
C> pip install IMDbPY.

2

Load the script `C:\labs\top250starter.py` into Pycharm. This script imports the IMDb module and creates an IMDb HTTP access class for searching for movie information. Compile and run the script and confirm the output displays the top 250 movies.

3

Modify the script so that the movie names are displayed alongside their ranking from 1 to 250. Ensure the ranking is right justified to 4 characters so that the `{:>4}` is aligned. For example:

1: The Shawshank Redemption  
2: The Godfather  
..  
99: North by Northwest

4

Using regular expressions, modify the script to prompt the user to enter their favourite movie and display its ranking, if found. Ensure that the case is ignored.

## Regular Expressions - stretch exercises for Kings

5

Copy the `top250.py` script to a new file called `search250.py` in the `C:\labs` folder.

6

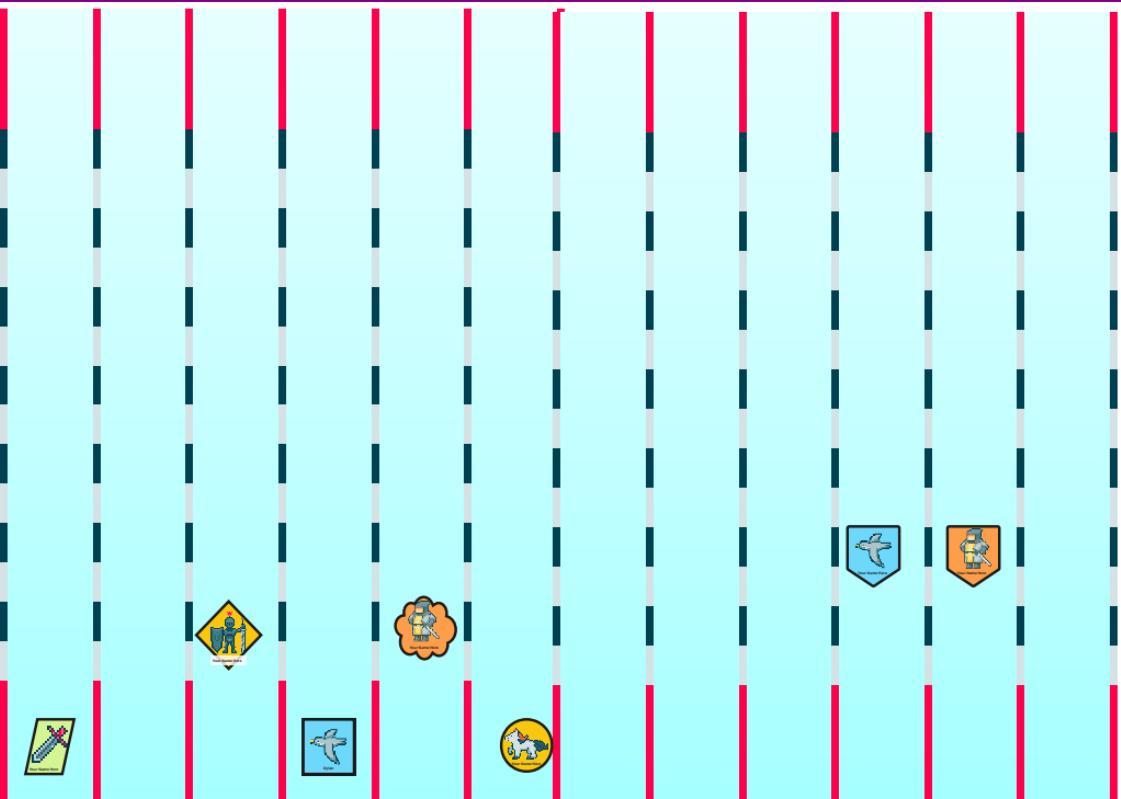
Modify the `search250.py` script so that it display the movies that match the following patterns and make a note of how many movies were matched for each. Ignore case for all matches.

- a. Movie names starting with 'the' (50)
- b. Movie names with a digit in the name (11)
- c. Movie names with 3 consecutive vowels (5)
- d. Movie names starting and ending with the same char (15)

7

Remember the `'C:\labs\display_unused_ports.py'` program from the previous lab which displays unused ports between 1 and 200? Modify the code to use Regular Expressions rather than string splitting to isolate and extract the port number.

Move your piece down as you complete the exercises



**Not  
confident**

**Quite  
confident**

**Very  
confident**

Back to  
the quest





# Functions

The Art of reusing named blocks of code

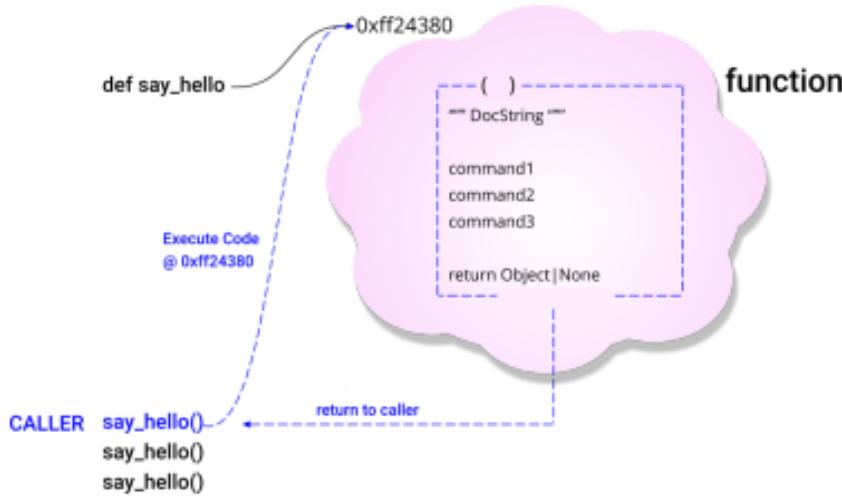


# CODE REUSE

# Functions

Earlier in the DIGITAL session, you learned about functions and how useful they are for making code sharable and reusable. You saw function definitions, passing parameters to a function, returning a value from a function, and documenting functions with docstrings.

You can do more with functions! You will learn about different types of parameters, variable scope, the idea of annotating functions, lambda functions, and using functions as objects. This includes passing a function as a parameter and returning a function from another function.



```
#!/bin/python
# Name:    demo_user_functions.py
# Author:  QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to
# define, name and execute (reuse) user defined functions.
"""
    Displays greeting messages.
"""

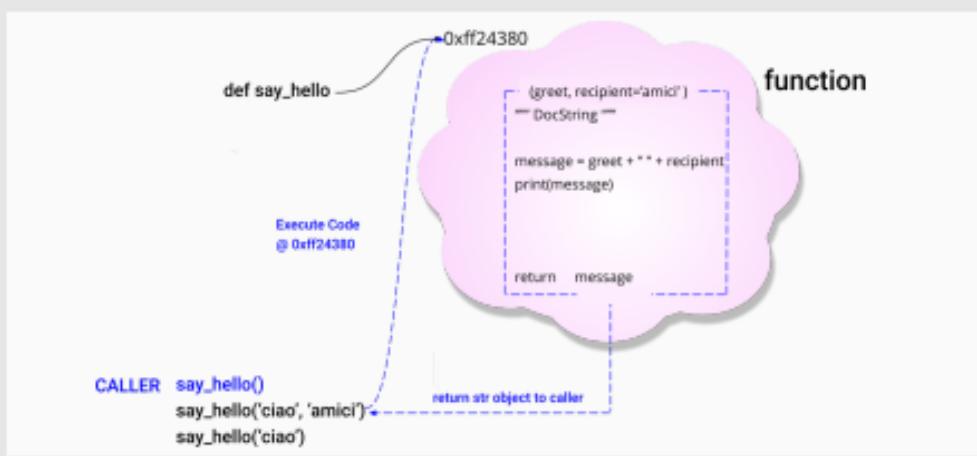
import sys

# Example of a user defined function
def say_hello ():
    """ Display the SAME GREETING to the SAME RECIPIENTS """
    message = "Hello" + " " + "World"
    print ( message )
    return None

def main ():
    """ Main function """
    say_hello () # CALLER - passes control to function say_hello().
    say_hello () # Can reuse function.
    say_hello ()
    say_hello ()
    return None

if __name__ == "__main__":
    main()
    sys.exit (0)
```

# Parameters



User functions can optionally accept a finite number of parameters.

- Formal parameter names are defined within the parentheses of function declaration
- Actual values are passed in the CALLER's parentheses
- Literal values are passed by COPY/VALUE, e.g. (100, 200, 'hello')
- Named objects are passed by Reference, e.g. (pattern, filename). So changing a parameter within a function will alter the caller's object. To prevent this for lists and dictionaries, pass a slice of the structure. e.g. function\_call(my\_list[:]). It's a hack!
- Parameters can be assigned defaults, but following parameters must also have defaults
- Parameters can be passed as positional, named or a mix (positional followed by named)

```
#!/bin/python
# Name: demo_user_functions_with_parameters.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to
# define, name, call and pass parameters in and return objects back.
"""

    Displays greeting messages.
"""

import sys

# Example of a user defined function with parameter passing
# and optional defaults
def say_hello(greeting="ciao", recipient="amici"):
    """Display a given greeting to a recipient"""

    # Convert parameters to strings for safety!
    message = str(greeting) + " " + str(recipient)
    print(message)

    # return None
    return message

def main():
    """Main function - say hello to our friends"""

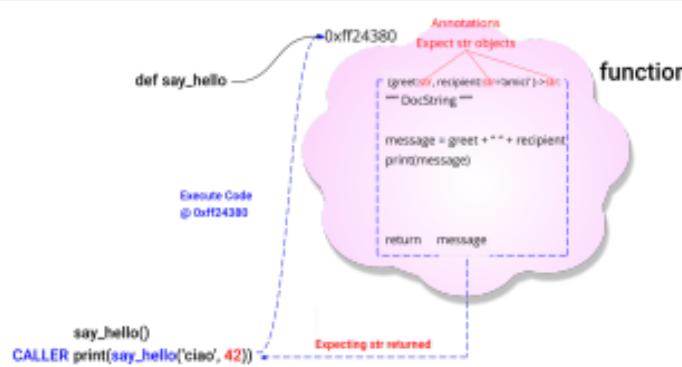
    say_hello("Hello", "world") # Positional parameter passing
    say_hello(greeting="Salutare", recipient="prietenii") # Named parameter passing
    say_hello("Namaste", recipient="meine dost") # Mixed parameter passing
    say_hello(recipient="mes amis", greeting="Bonjour") # Mixed in different order
    say_hello("Nazdar", 42) # Try passing in an Integer!
    say_hello()

    # If the function returns an object we can then have the CALLER as
    # a r-value. That means on the RHS of an assignment or embedded within
    # a larger expression.
    # num = say_hello()
    print(f"Default message is {say_hello()}")


    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# Annotations



Function annotations were introduced in PEP 3107. They are used to provide additional information about the function, including what data types are expected for the input parameters and return value. They are like an embedded comment, but are more than a comment (which are ignored by the compiler). They are preserved in the function attribute `__annotations__`.

Annotations have no meaning except 'commenting in-code' of what types of data are hoped for. But 3rd party modules might use them for additional docstrings or for enforcing data type on parameters or return values. Python will remain a dynamically typed language!

```
#!/bin/python
# Name: demo_user_functions_annotations.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to
# define, name, call and pass parameters in and return objects back.

# Displays greeting messages.
import sys

# Example of a user function ANNOTATIONS (Py3.5, not enforced)
def say_hello( greeting : str ="ciao" , recipient : str ="amici" )-> None:
    """ Display a given greeting to a recipients """
    # Annotations not enforced by Python so convert parameters
    # into strings for safety.
    message = str(greeting) + " " + str(recipient)
    print(message)
    return None

def main():
    """ Main function - say hello to our friends """
    say_hello("Hello", "world") # Positional parameter passing.
    say_hello(greeting="Salutare", recipient="prietenii") # Named parameter passing.
    say_hello("Namaste", recipient="mere dori") # Mixed parameter passing.
    say_hello(recipient="mes amis", greeting="Bonjour") # Mixed in different order.
    say_hello("Nazdar", 42) # Try passing in an integer!
    say_hello()

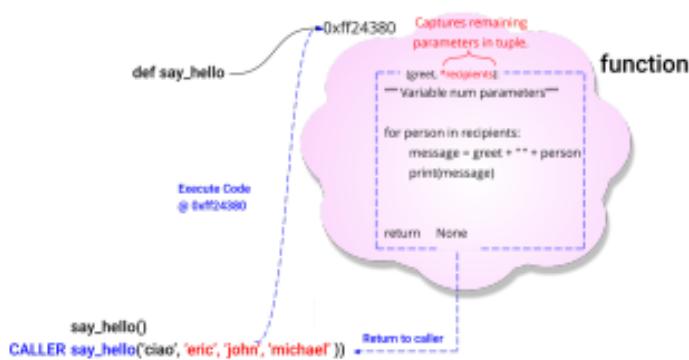
# If the function returns an object we can then have the CALLER as
# a r-value. That means on the RHS of an assignment or embedded within
# a larger expression.
# num = say_hello()
print(f"Default message is {say_hello()}" )

# Attributes of functions can be displayed including annotations.
print(f"Annotations for say_hello() = {say_hello.__annotations__}")

return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# Variadic Functions



**A Variadic function** can accept a variable number of parameters. For example, the built-in function `print()` can accept any number of parameters. User functions can also accept variable number of parameters into either a **tuple** with a `**` prefix or into a **dictionary** with a `***` prefix. This works in a similar way to unpacking as discussed with collections.

**DEFINITION:** `def display_vat(**kwargs)`

**CALLER:** `display_vat(vatpc=15, gross=9.55, message='Summary')` # Use named parameters when passing to a dict.

```
#!/bin/python
# Name: demo_user_functions_variadic.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to
# define, name, call and pass parameters in and return None.
...
    Displays greeting messages to ALL friends and foes.
...
import sys

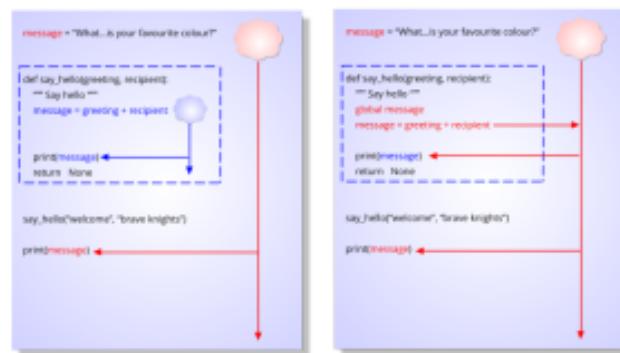
# Example of a Variadic function.
def say_hello ( greeting = "ciao" , * recipients ):
    ...
        Accept a greeting string and variable number of recipients and
        display message to ALL recipients and returns None.
    ...
        # Use an iterator for loop to iterate through objects in tuple.
        for person in recipients :
            message = str ( greeting )+ " " + str ( person )
            print ( message )
        ...
            # return None
            return None

def main ():
    """ Main function - say hello to our friends """
    ...
        # Pass in a variable number of parameters to the function.
        say_hello ( 'None shall pass' , 'green knight' )
        say_hello ( 'None shall pass' , 'green knight' , 'arthur' , 'patsy' )
    ...
        return None

if __name__ == "__main__":
    main()
    sys.exit ( 0 )
```

# Scope

## Local Vs Global



**Scope** defines the **life** and **visibility** of a variable/object, in other words, what part of the code can see and use it. Python supports global scope and function/local scope. Global variables/objects are visible to the entire program including nested functions.

But if a value is assigned in the function before it is used, then it becomes a local variable (visible and lives for duration of function).

To change the value of an outer global variable within a function (frowned upon), declare the variable as **global**.

```
#!/bin/python
# Name: demo_user_functions_scope.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate the scope (life + visibility)
# of variables/objects
*** Displays greeting local and global messages.
*** import sys
message = "What...is your favourite colour?" # Example of a user defined function with parameter passing
# and optional defaults
def say_hello ( greeting = "welcome" , recipient = "brave knights" ):
    """ Display a given greeting to a recipients """
# global message # Try executing with and without this line commented
# An assignment to message forces Python to create a new local object.
message = str ( greeting ) + " " + str ( recipient )
print ( message )
# return None
return message
def main():
    """ Main function - say hello to our friends """
    say_hello ()
    print ( f "function return is {say_hello ()}" )
    print ( message )
# return None
if __name__ == "__main__":
    main()
    sys.exit ( 0 )
```

# Docstrings

## HELP, I'm lost!

So have you been wondering why we have been putting tripled quoted strings in our programs and functions?

Well these are called DocStrings and usually appear immediately after the definition of a program/module, class, function or method.

Docstrings can be simple one-liners or multi-line and should not be descriptive (that is for # comments). Rather they must follow "Do this, return that" format. Multi-line docstrings can be more elaborate, and are described in PEP 257, and can include details about input parameters and return values. Docstrings for classes (later in LIVE) should summarise its behaviour and list public methods. The `__doc__` attribute can be used to access docstrings.

And they are read by the Python `help()` function - so now you know where we are getting help from for our modules and functions!

```
#! bin/python
# Name: demo_user_functions_docstrings.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate another example of
# creating functions with DocStrings, parameter passing, return values.
"""
Calculator program with Add, Divide and Multiply functionality.
"""

import sys

def multiply ( x, z):
    """
    Accepts two numeric parameters and return product.

    param x (int/float): Integer or float
    param z (int/float): Integer or float
    return (int/float): Numeric product of x and z
    """
    return x * z

def add ( x, z):
    """
    Accepts two numeric parameters, adds them and returns sum.

    param x (int/float): Integer or float
    param z (int/float): Integer or float
    return (int/float): Numeric of x summed with z
    """
    return x + z

def divide ( x, z):
    """
    Accepts two numeric parameters, divides first by the second and returns result.

    param x (int/float): Integer or float
    param z (int/float): Integer or float
    return (str): 1-string of x divided by z to 3 decimal places.
    """
    return f" { (x/z):.3f} "

def main ():
    """
    Single Line doc-strings are ideal if the function is self-explanatory """
    print ( f"4 * 3 = { multiply ( 4, 3 ) }")
    print ( f"4 + 3 = { add( 4, 3 ) }")
    print ( f"4 / 3 = { divide ( 4, 3 ) }")

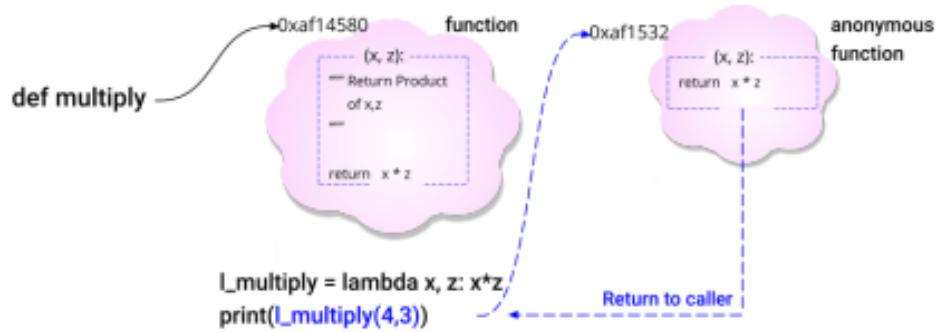
    """
    An example of a docstring for the built-in print function
    # print(print.__doc__)
    An example of a docstring for the sys module.
    # print(sys.__doc__)
    Displaying the docstring for user functions.
    print ( multiply.__doc__ )

    return None

if __name__ == "__main__":
    main()
    sys.exit ( 0 )
```

# Lambda

## Functions



```
#!/bin/python
# Name: demo_user_functions_lambda.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate another example of
# creating functions using lambda (anonymous) functions.
# Simple one-line docstrings are used.
"""
Calculator program with Add, Divide and Multiply functionality.
"""

import sys

def multiply ( x, z):
    """ Multiply two numeric parameters and return result """
    return x * z

def add ( x, z):
    """ Add two numeric parameters and return result """
    return x + z

def divide ( x, z):
    """ Divide two numeric parameters and return formatted str. """
    return f" {x/z:.3f} "

def main():
    """ This is the main program function """
    print ( f"4 * 3 = {multiply ( 4, 3)}")
    print ( f"4 + 3 = {add ( 4, 3)}")
    print ( f"4 / 3 = {divide ( 4, 3)}")

    # A lambda function is a small anonymous function that can take
    # multiple arguments but only has one expression. And is an alternative
    # way to defining simple functions without the 'def' statement.
    l_mul = lambda x, z: x * z
    l_add = lambda x, z: x + z
    l_div = lambda x, z: f" {x/z:.3f} "

    print ( "-" * 20)

    print ( f"4 * 3 = {l_mul ( 4, 3)}")
    print ( f"4 + 3 = {l_add ( 4, 3)}")
    print ( f"4 / 3 = {l_div ( 4, 3)}")
    return None

if __name__ == "__main__":
    main()
    sys.exit (0)
```

# Nested Functions

## Nested function

```
def outer():
    num = 42
    def inner():
        print("inner, num = {num}")
    inner()
    print("Outer, num = {num}")
    return None

outer()
inner()
```

## Variables in nested functions

```
knight = "King Arthur"           ← global variable

def brave_knights():
    knight = "Sir Robin"          ← local variable
    def who_goes_next():
        nonlocal knight
        knight = "Sir Lancelot"
        return None
    who_goes_next()
    print("inner, num = {num}")
    return None

brave_knights()
print("({knight}) will go next")
```

Python allows the nesting of functions inside each other, which not all languages support. It can be useful for simple reusable, recursive functions, function factories (a function can be returned from a function like any object) or for closures.

Like nested objects, functions follow the same scope rules. Nested functions can only be called within the function they are defined.

Python has a clean, safe, and dynamic way of managing variables/objects. As soon as you make an assignment, a new object is created - either a **GLOBAL** or a **LOCAL** (within a function). This mostly works well, but on occasion we might have nested functions, and we don't quite want to create a new variable and don't want to access the global variable either. This is where the **nonlocal** keyword proves useful to indicate the variable is referring to an enclosing (outer) scope variable.

```
#!/bin/python
# Name:    demo_user_functions_nonlocal.py
# Author:  QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate that Python supports nested
# functions global, local and not-so local variables.
"""
    Nested functions and scope.
"""
import sys

knight = "King Arthur"

def who_goes_next():
    """
        Who is next to walk the bridge of death?
    """
    knight = "Sir Robin" # Really?
def brave_knights():
    nonlocal knight
    knight = "Sir Lancelot" # Of course, Lancelot is far braver!
    return None
brave_knights()
print(f"({knight}) will go next")
return None

def main():
    """
        Who goes there?
    """
    who_goes_next()
    print(f"({knight}) will go next")
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```



# Quiz

## User Functions Knowledge Check



5-10 mins

### 1. What does (\*, ) at the beginning of the parameter list do?

```
*, trailing parameters are not "positional" and must be named
def my_function(x, *, y):
    do stuff

my_function(x = 10, y = 20)
my_function(2, y=4)
my_function(x=10, 4) # BANG
```

### 2. What is the output of the following function call?

```
>>> def my_func(num):
>>>     return num + 10
>>>
>>> my_func(10)
>>> print(num)
```

BANG (on print(num) line)

### 3. True or False: Python functions always return a value

True

(Python functions always return None unless explicit value)

### 4. What is the output of the following function call?

```
>>> def display_info(**kwargs):
>>>     for z in kwargs:
>>>         print(z)

>>> display_info(movie="braveheart", year="1995")
```

movie  
year

to print keys and values use:  
def display\_info(\*\*kwargs):  
 for k, v in kwargs.items():  
 print(k, v)

# Exercises

⌚ 35 mins

## Functions - exercises for 'brave' knights

1

Write your own calculator program called 'C:\lab\calc.py' with add, divide, multiply, subtract and modulus functions.

Test your functions by passing in the following parameters.

```
add(11, 5)  
divide(11, 5)  
multiply(11, 5)  
subtract(11, 5)  
modulus(11, 5)
```

2

You did add Docstrings to your functions? If not, add appropriate Docstrings to each function. To test, start IDLE and load your script (File/Open), then run (<F5>). Then in Python Shell type:

```
>>> help(add)  
>>> help(multiply)
```

3

Modify the calc.py script to allow the add and multiply functions to accept a variable number of parameters. Test the modified functions by passing in the following parameters.

```
add(11, 5, 33, 15, 6.5)  
multiply(11, 5, 33, 15, 6.5)
```

## Functions - stretch exercises for Kings

4

In the following exercises, you will use existing demonstrations and code, and improve by creating and using reusable functions.

Open the 'C:\labs\search250.py' script you created in the Regex section. Modify the script so that it has a reusable function called search\_movie() which accepts one parameter - the Regex pattern to search. Test the function by calling it from the main() function with the same patterns from the previous lab exercises.

5

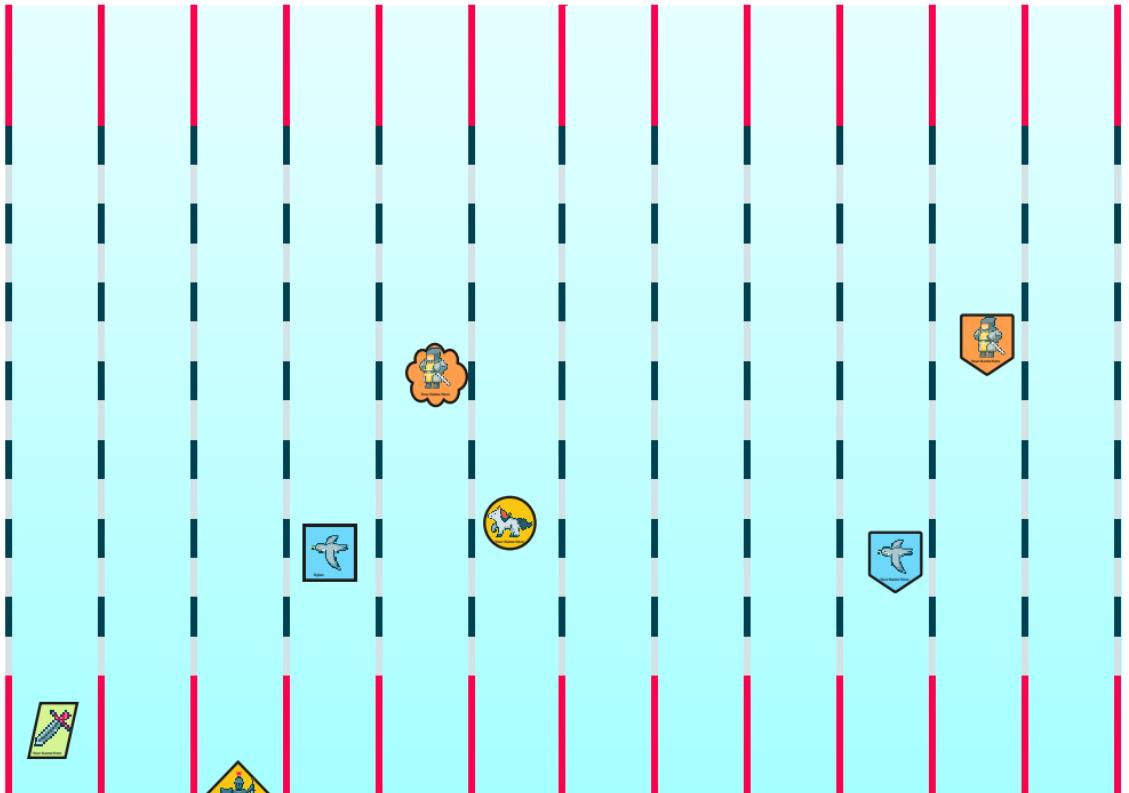
Open the 'C:\labs\searchWords.py' script. It searches for regular expressions patterns in the file 'C:\labs\words' and displays them to the Python shell. Modify the script and create a variadic function called search\_pattern() which can accept one regex pattern followed by one or more files.

a. If no files are given, then default to 'C:\labs\words'.  
b. Iterate through each file printing out lines that match.

Test with the following function calls in the main() function:-

```
search_pattern(r"^[A-Z].*1$")  
search_pattern(r"^(A-Z).*1$", r"C:\labs\words", r"C:\labs\words")
```

Move your piece down as you complete the exercises



**Not  
confident**

**Quite  
confident**

**Very  
confident**

Back to  
the quest





# Advanced Collections

Interesting and cool things you can do with collections



## COLLECTIONS

1. filtering
2. lambda
3. comprehensions
4. generators
5. copying



# FILTERING

Source Collection



Optional Filtering



New Collection



```
#! /usr/bin/python
# Name: demo_collections_comprehensions.py
# Author: Q/A2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to filter collections using
# the built-in filter() function, lambda functions and comprehensions.
#--#
# Filter collections of cities into warm cities in Celsius.
#--#
import sys
weather = { 'Glasgow' : 11,
            'London' : 21,
            'Edinburgh' : 15,
            'Manchester' : 18,
            'Thurso' : 9
        }

def filter_city(city):
    """ Filter cities by temp """
    if weather[city] >= 15:
        return True
    else:
        return False

def main():
    """ Filter collection using filter(), lambda and comprehensions """
    # For Loop plus source collection, optional if condition, and expression.
    warm_cities = {}
    for city in weather: # Iterator for loop + collection.
        if weather[city] >= 15: # Optional condition (filtering)
            warm_cities[city] = weather[city] # Expression
    print ("*1.Warm Cities = " + (str(warm_cities) + "\n"))

    # For Loop plus source collection, optional if condition with user function, and expression.
    warm_cities = {}
    for city in weather:
        if filter_city(city):
            warm_cities[city] = weather[city]
    print ("*2.Warm Cities = " + (str(warm_cities) + "\n"))

    # Built-in filter() function plus source collection, user function for filtering.
    warm_cities = list(filter(filter_city, weather))
    print ("*3.Warm Cities = " + (str(warm_cities) + "\n"))

    # Built-in filter() function plus source collection, lambda function for filtering.
    wee_names = list(filter(lambda city: weather[city] >= 15, warm_cities))
    print ("*4.Warm Cities = " + (str(wee_names) + "\n"))

    # For Loop plus source collection, optional if condition, and expression. LIST comprehension.
    warm_cities = [ city for city in weather if weather[city] >= 15 ]
    print ("*5.Warm Cities = " + (str(warm_cities) + "\n"))

    # For Loop plus source collection, optional if condition, and expression. LIST comprehension.
    warm_cities = [ city for city in weather if weather[city] >= 15 ]
    print ("*5.1.Warm Cities = " + (str(warm_cities) + "\n"))

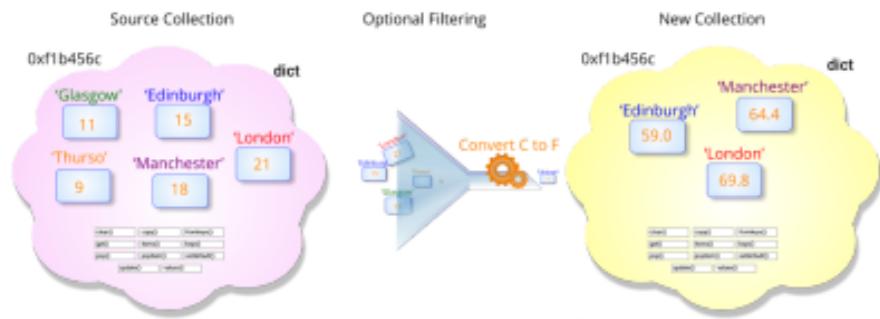
    # For Loop plus source collection, optional if condition, and expression. DICT comprehension.
    warm_cities = { city : weather[city] for city in weather if weather[city] >= 15 }
    print ("*5.2.Warm Cities = " + (str(warm_cities) + "\n"))

    # For Loop plus source collection, optional if condition, and expression. SET comprehension.
    warm_cities = { city for city in weather if weather[city] >= 15 }
    print ("*5.3.Warm Cities = " + (str(warm_cities) + "\n"))

    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# FILTERING +



```

# bin/python
# Name: demo_collections_comprehensions_plus_c2f.py
# Author: QA2.0, Donald Cerman
# Revision: v1.0
# Description: This program will demonstrate how to filter collections using
# the built-in filter() function, lambda functions and comprehensions.

# Filter collections of cities into warm cities in fahrenheit.

import sys

weather = { 'Glasgow' : 11,
            'London' : 21,
            'Edinburgh' : 15,
            'Manchester' : 18,
            'Thurso' : 9 }

def filter_city(city):
    """ Filter cities by temp """
    if weather[city] >= 15:
        return True
    else:
        return False

def c2f(temp):
    """ Accept temp in celsius and return temp in fahrenheit """
    return (temp * 9/5) + 32

def main():
    """ Filter collection using filter(), lambda and comprehension """
    # For Loop plus source collection, optional if condition, and expression
    warm_cities = []
    for city in weather: # Iterator for loop + collection
        if filter_city(city):
            warm_cities.append(c2f(weather[city])) # c2f Expression
    print ("#1.Warm Cities= ", warm_cities)

    # For Loop plus source collection, optional if condition with user function, and expression
    warm_cities = []
    for city in weather:
        if filter_city(city):
            warm_cities.append(c2f(weather[city]))
    print ("#2.Warm Cities= ", warm_cities)

    # Built-in filter() function plus source collection, user function for filtering
    warm_cities = list(filter(filter_city, warm_cities))
    print ("#3.Warm Cities= ", warm_cities)

    # Built-in filter() function plus source collection, lambda function for filtering
    warm_cities = list(filter(lambda city: weather[city] >= 15, warm_cities))
    print ("#4.Warm Cities= ", warm_cities)

    # For Loop plus source collection, optional if condition, and expression. LIST comprehension.
    warm_cities = [c2f(weather[city]) for city in weather if filter_city(city)]
    print ("#5.Warm Cities= ", warm_cities)

    # For Loop plus source collection, optional if condition, and expression. LIST comprehension.
    warm_cities = [c2f(weather[city]) for city in weather if filter_city(city)]
    print ("#5.1.Warm Cities= ", warm_cities)

    # For Loop plus source collection, optional if condition, and expression. DICT comprehension.
    warm_cities = { city : c2f(weather[city]) for city in weather if filter_city(city)}
    print ("#5.2.Warm Cities= ", warm_cities)

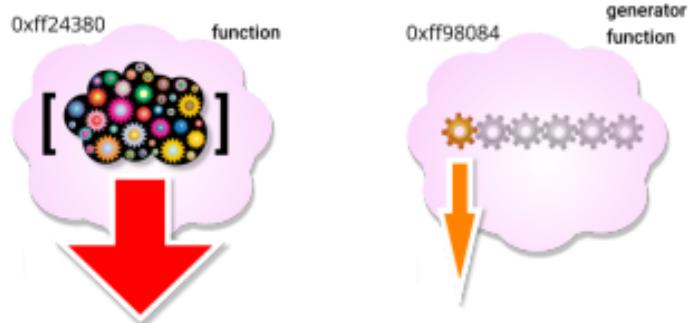
    # For Loop plus source collection, optional if condition, and expression. SET comprehension.
    warm_cities = { city for city in weather if filter_city(city)}
    print ("#5.3.Warm Cities= ", warm_cities)

    return None

if __name__ == '__main__':
    main()
    sys.exit(0)

```

# GENERATING



```
#!/bin/python
# Name: demo_collections_generator_functions.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to generate a collection of
# numbers using generator functions.
"""
Generate and display collection of numbers lazily...
"""

import sys

def get_numbers():
    """Return an ENTIRE list of numbers"""
    print ("Executing get_numbers().")
    numbers = []
    for x in range (0, 10):
        numbers.append (x)
    return numbers

def generate_numbers():
    """GENERATE one object/number at a time = Lazy List"""
    print ("Executing generate_numbers().")
    for x in range (0, 10):
        yield x

def generate_numbers2():
    """GENERATE one object/number at a time = Lazy List
    using a return and list comprehension, oink"""
    print ("Executing generate_numbers2().")
    return [ x for x in range (0, 10) if x%2==0 ]

def main():
    """Generate collection of numbers"""

    # Try each of these with large numbers for range()
    for x in get_numbers():
        print (x)

    for x in generate_numbers():
        print (x)

    for x in generate_numbers2():
        print (x)

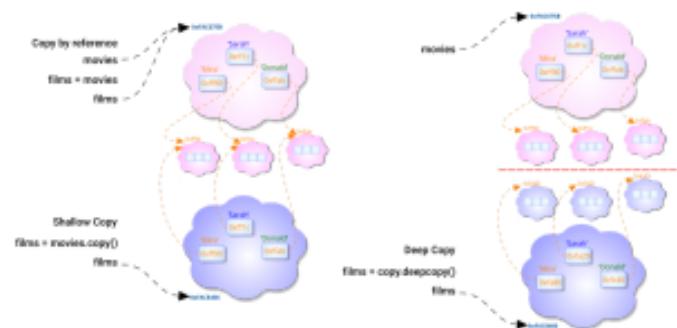
    # Alternative ways to use a generator function and access a lazy
    # list using the builtin next() function and combined with a while loop.
    gen = generate_numbers()
    while True:
        num = next (gen, -1)
        if num == -1:
            print (num)
        else:
            break

    # Alternatively, manually get the next() object...
    gen = generate_numbers()
    num1 = next (gen, False)
    num2 = next (gen, False)
    num3 = next (gen, False)

    print (num1, num2, num3)
    return None

if __name__ == "__main__":
    main()
    sys.exit (0)
```

# COPYING



```
#!/bin/python
# Name:          demo_collections_copying.py
# Author:        QA2.0, Donald Cameron
# Revision:     v1.0
# Description:  This program will demonstrate how to copy collections
## by memory reference, shallow copy and deep copy.

"""
    Copying 2_Collections.
"""

import sys
import pprint
import copy

def main():
    """
        Copy 2_Collections """
    movies = {'Donald': ['Dracula', 'Deliverance', 'Descendants'],
              'Mira': ['Matrix', 'Mad Max', 'Magnolia'],
              'Sarah': ['Seven', 'Scream', 'Saving Private Ryan']}
    # Comment each of the following on at a time!
    # films = movies # Copy by memory reference
    # films = movies.copy() # Shallow Copy, nested objects shared.
    # films = copy.deepcopy(movies) # Deep copy to all levels.

    movies['Mira'][1] = 'Magnificent Seven'
    movies['Brian'] = ['Braveheart', 'Brave', 'Babe']

    pprint pprint(movies)
    print("-" * 60)
    pprint pprint(films)
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```



# Quiz

Advanced Collections Knowledge Check



5-10 mins

**1. What type of comprehension is this, and what would it return?**

```
>>> [ num for num in range(0, 10) if num % 2 == 0 and num % 4 == 0 ]
```

**To be revealed**

**2. How many Gumbies (might have to google that!) would be displayed?**

```
>>> gumbys = ['eric', 'michael', 'terry', 'john', 'terry', 'graham']  
>>> { name for name in gumbys if len(name) <= 5 }
```

**To be revealed**

**3. Would this sequence of commands return True or False?**

```
>>> movies = {'Cruella': '2021', 'Mulan': '2020', 'Mary Poppins Returns': '2018'}  
>>> films = movies.copy()  
>>> movies['Mulan'] = 1998  
>>> films == movies
```

**To be revealed**

# Exercises

⌚ 35 mins

Move your piece down as you complete the exercises



## Advanced Collections - exercises for 'brave' knights

You should be familiar with the built-in `range()` function that generates a sequence of integers from a start point to a stop point with an optional step increment.  
`range(stop) # default start=0, step=1`  
`range(start, stop, step)`

Unfortunately, it only works with integers. In this exercise, we will create our own simple version of the built-in `range()` function but for floating point numbers. Create a new script 'C:\labs\gen.py' with a generator function called `frange()` which accepts at least two parameters (start, stop) and an optional parameter with default (step=0.25). Be wary of the possibility of a step = zero being passed in!

`frange(start, stop, step=0.25)`

Test with the following calls in `main()`:

```
print(list(frange(1, 1, 3)))
print(list(frange(1, 3, 0.33)))
print(list(frange(1, 3, 1))) # Should print [1.0, 2.0].
print(list(frange(3, 1))) # Should print an empty list.
print(list(frange(1, 3, 0))) # Should print an empty list.
print(list(frange(1, -0.5, 0.1)))
print(frange(1, 2)) # Should print <generator object frange at 0x...>
```

`for num in frange(3.142, 12):`

Modify your 'C:\labs\gen.py' script and enhance your `frange()` function so that it can accept only one parameter that acts as the stop of the sequence. The starting value for the range should then default to 0, with step set to 0.25.

Test the function with the following code in `main()`:

```
result1 = list(frange(0, 3.5, 0.25))
result2 = list(frange(3.5))
if result1 == result2:
    print("Default worked!")
else:
    print("Oops! (result1) not equal to (result2)")
```

## Advanced Collections - stretch exercises for Kings

Did you notice any inaccuracies in the numbers in the previous exercises? This is expected and is the nature of using floating points. They are so serious that this code should not be used in a production environment. A solution is to use the decimal module from the Python Standard library. This has a `decimal.Decimal` class constructor that converts integers/strings to Decimal objects - do this for all the parameters (start, stop and step) AND then convert the value to be yielded back to a float.

Modify your script 'C:\labs\gen.py' and `frange()` function to use the decimal module. The test results should be more sensible! Test using same calls in question 4.

**Not  
confident**

**Quite  
confident**

**Very  
confident**

[Back to  
the quest](#)





# Packages, Modules, Namespaces

Let's take CODE REUSE up a level

⌚ Modularity

⌚ Building a Package

⌚ Is is a program or a module?

⌚ Namespaces - one honking great idea!

⌚ The `_main_` namespace trick

⌚ Profiling

⌚ Doc Testing

⌚ Unit Testing



# Modules & Packages

**Modular programming** is the process of breaking down a large complex problem into separate, smaller and more manageable subtasks or modules. These simpler individual modules can then be completed (possibly in parallel for larger teams) and then glued back together like building blocks.

There are several advantages to using a modularised approach including:

- **Simplicity:** a module typically focuses on one small part of the problem and is easier to design and code.
- **Maintainability:** modules are independent and designed, coded, debugged and tested and modified with impacting others.
- **Reusability:** modules can be reused across projects and programs. The 'Holy Grail'!
- **Scoping:** modules can have a separate namespace (honest great idea) avoiding conflicts with identifier names.

Python supports **Packages**, **Modules** and **Functions** that all promote the concept of modularisation.

There are three ways to define a module to be used in Python:

1. It can be written in Python itself with a .py suffix. This is the most common and the one we will focus on.
2. It can be written in C and loaded dynamically at run-time. The re module uses this!
3. A built-in module is intrinsically contained with the compiler, e.g. built-ins.

## Python Package

Technically, a Python Package is just another module with sub-modules, sub-packages and a `__path__` attribute. So importing a module or package is really just the same. But if you are looking for a definition - then a regular Package is NAMED directory with a logical group of files, modules, sub-folders AND a file called `__init__.py`. If the `__init__.py` is missing then it's called a Namespace package.

In general, sub-modules and sub-packages are not imported when a package is imported. The `__init__.py` can be used to include all or any sub-modules or sub-packages. It can also contain global constants and functions and can also be empty!

Python automatically stores compiled python modules in a `__pycache__` folder which is found in the Package directory or Python install directory. When you are importing a module, including Python Standard Library modules, it is the pre-compiled modules with a suffix .pyc which are loaded, reducing the time to compile. If the .py module file has a newer timestamp then it is re-compiled and copied back in to the `__pycache__` folder.

## Finding Modules

The Python interpreter finds the modules - as a built-in, in the current or package directory, in a list of directories defined in the `PYTHONPATH` environment variable, or an installation-dependent list of directories configured during Python install.

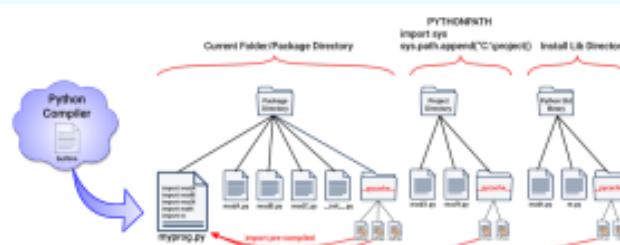
On Linux, `$ export PYTHONPATH="${PYTHONPATH}:$(HOME)/python/lib:/projectX/lib"`  
On Windows, [Right Click] Computer > Properties > Advanced System Settings > Environment Variables > System Variables > New

Within a program,

```
>>> import sys  
>>> sys.path.append(r"C:\labs\projects\Proj_X")  
>>> print(sys.path)
```

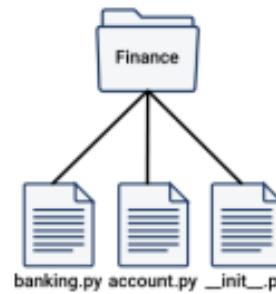
Importing a Package, and then a sub-package (from the current package),

```
>>> import my_package  
>>> from . import my_sub_package
```



# Creating Package

**Packages** are a logical group of modules that share the same directory and namespace and help avoid naming collisions using the **dot notation** - in a similar way that hierarchical file systems can have files with the same names but with a different pathname. Packages can be created and named in Pycharm, and a new directory will be created with the `__init__.py`.



```
#!/bin/python
# Name: banking.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This module defines several functions for banking app.

"""
Manages Bank accounts with deposits and withdrawals.
"""

import sys

def deposit (balance):
    """
    Deposit monies into bank account
    """
    amount = float (input ("Please enter amount to be added: "))
    balance += amount
    print (f"Deposited {amount} ")
    return balance

def withdraw (balance):
    """
    Withdraw monies from bank account
    """
    amount = float (input ("Please enter amount to withdraw: "))
    balance -= amount
    print (f" {amount} withdrawn. ")
    return balance

def main ():
    """
    Withdraw and deposit monies
    """
    bank_balance = 0
    print (f"Welcome to mBA (mini Bank Account) App" )

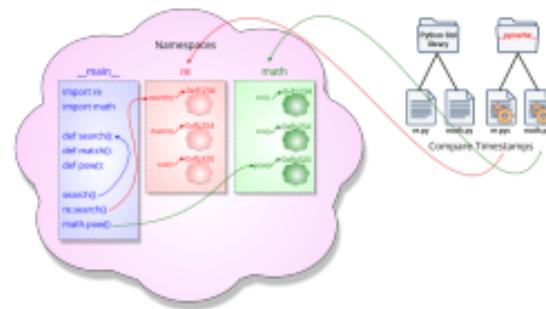
    while True:
        menu = f"""
mBA Menu
-----
Current Balance {bank_balance}
1. Deposit monies.
2. Withdraw monies.

"""
        print (menu)
        option = input ("Enter option (1-2, q=quit): ")
        if option == "1":
            bank_balance = deposit (bank_balance )
        elif option == "2":
            bank_balance = withdraw (bank_balance )
        elif option .lower ()== "q":
            break
        else :
            print ("Invalid option" )

    print ( "Thanks for using the mBA app." )
    return None

if __name__ == "__main__":
    main()
    sys.exit (0)
```

# Packages & Namespaces



```
#!/bin/python
# Name: account.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This is MBA - Minions Banking App used by millions of minions!
```

```
"""
Withdraw, Deposit and Display banking information.
"""


```

```
import sys
import banking
# from bank_account import * # TRY THIS!

def display_info ( bal , scode ):
    """ Display bank account info """
    print ( f"Current balance = £ { bal :.2f} , sort code = { scode }" )
    return None

def main ():
    """ Withdraw, deposit and display bank account information """
    bank_balance = 34_500.23
    sort_code = "80-45-37"


```

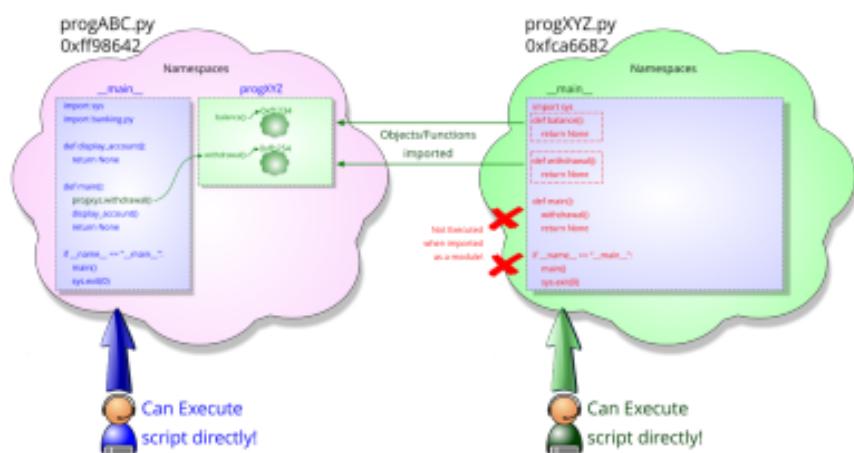
```
    print ( "Welcome to MBA - Minions Banking App" )
    name = input ( "Enter your name: " ). title ()
    while True :
        menu = f"""
        Menu Options
        1. Display Balance
        2. Deposit monies
        3. Withdraw monies
        """
        print ( menu )
        option = input ( "Enter option (1-3, q=quit): " )
        if option == "1" :
            display_info ( bank_balance , sort_code )
        elif option == "2" :
            bank_balance = banking .deposit( bank_balance )
        elif option == "3" :
            bank_balance = banking .withdraw( bank_balance )
        elif option .lower () == "q" :
            break
        else :
            print ( "Invalid option" )

    print ( f"Thank you { name } for using our app!" )
    return None
```

```
if __name__ == "__main__":
    main()
    sys .exit ( 0 )
```

# Namespace Trick

Program == Module: The Namespace Trick



Have you noticed the `__name__ == "__main__"` test we have been using in conjunction with a `main()` function? It's called the namespace trick and allows your script to act as a module and your module to act as a script. In other words, it allows the user to execute the module directly as a script, and also import the module into another script without executing the `main()` function!

It's also part of structuring our python scripts correctly. Continue using it! Oh, and always end your script with an error code, 0 indicates zero errors and any other integer (1-255) indicates an error code! This exit code is passed back to the calling program and indicates success or failure of your script.

## Importing - the different ways

`import bank_account  
bank_account.deposit()`

Safer and traceable

`import sys, re, bank_account`

PEP8 disapproves!

`import sys  
import re  
import bank_account`

PEP8 approves!

`import bank_account as b_acc  
b_acc.deposit()`

Module alias

`from bank_account import *`

 Namespace Collision

`from bank_account import deposit`

Safer, be careful!

`from bank_account import (deposit as dp)  
dp()`

Confusing

# Profiling

## Performance & Quality

```
#!/usr/bin/python
# Name: demo_profiling_bench.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to benchmark/profile code
# to improve performance and quality using the cProfile module
# Search a given file using regular expressions and display
# matched lines. Analyse performance of code change,
# import sys
# import re

def search_pattern(pattern, file="C:\lab\words.txt"):
    """Display lines in a given file which match a Regular Expression"""
    f_in = open(file, mode="rt")
    for line in f_in:
        m = re.search(pattern, line) # Return None or RE Match object
        if m:
            print(line, end="")
    f_in.close()
    return None

def main():
    """Demonstrate different 4 Regular Expression patterns"""
    search_pattern(r"\w{19,19\$}") # Match lines with exactly 19 chars.
    return None

if __name__ == "__main__":
    import cProfile
    cProfile.run("main()") # Display stats to console.
    #cProfile.run("main()", "stats.prof") # Write stats to file.
    sys.exit(0)
```

.analyse before

```
#!/usr/bin/python
# Name: demo_profiling_after.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This program will demonstrate how to benchmark/profile code
# to improve performance and quality using the cProfile module
# Search a given file using regular expressions and display
# matched lines. Analyse performance of code change,
# import sys
# import re

def search_pattern(pattern, file="C:\lab\words.txt"):
    """Display lines in a given file which match a Regular Expression"""
    f_in = open(file, mode="rt")
    reobj = re.compile(pattern)

    for line in f_in:
        g = reobj.search(line) # Return None or RE Match object
        if g:
            print(line, end="")
    f_in.close()
    return None

def main():
    """Demonstrate different 4 Regular Expression patterns"""
    search_pattern(r"\w{19,19\$}") # Match lines with exactly 19 chars.
    return None

if __name__ == "__main__":
    import cProfile
    cProfile.run("main()") # Display stats to console.
    #cProfile.run("main()", "stats.prof") # Write stats to file.
    sys.exit(0)
```

.analyse after

The Python Standard Library module '**cProfile**' can be used to understand the way your code behaves and is especially useful with large and complex programs. In its simplest form, the output generated will tell you how many calls each of your functions and methods received, and how long was spent executing each function (by line number and file/module name).

The analysis can be displayed to screen or written to a statistics file using a second parameter for the `cProfile.run()` function.

To analyse the statistics file:

```
C:\labs> python -m pstats stats.prof
% stats
```

# Doc Testing

## Automated Doc testing

The **doctest** module searches for text that looks like interactive Python sessions in the Docstrings, and then executes those sessions to verify that they work as shown. This is useful for checking that a module's Docstrings are up to date, for regression testing when code changes and to embed a tutorial within the documentation.

Test in interactive Python session.

```
>>> add(4, 3)
```

```
7
```

Copy and paste into function Docstring.

```
#!/bin/python
# Name:      demo_doctesting.py
# Author:    QA2.0, Donald Cameron
# Revision:  v1.0
# Description: This program will demonstrate another example of
# creating functions with DocStrings, parameter passing, return values
# ...
# Calculator program with Add, Divide and Multiply functionality.
# ...
import sys

def multiply ( x, z):
    """Accepts two numeric parameters and return product.
    >>> multiply(4, 3)
    12

    param x (int/float): Integer or float
    param z (int/float): Integer or float
    return (int/float): Numeric product of x and z
    """
    return x * z

def add ( x, z):
    """Accepts two numeric parameters, adds them and returns sum.
    >>> add(4,3)
    7

    param x (int/float): Integer or float
    param z (int/float): Integer or float
    return (int/float): Numeric sum of x summed with z
    """
    return x + z

def divide ( x, z):
    """Accepts two numeric parameters, divides first by the second and returns result.
    >>> divide(4, 3)
    '1.333'
    >>> divide(4, 0)
    Traceback (most recent call last):
    ZeroDivisionError: division by zero

    param x (int/float): Integer or float
    param z (int/float): Integer or float
    return (str): f-string of x divided by z to 3 decimal places.
    """
    return f'{x/z:.3f}'

def main():
    """Single Line docstrings are ideal if the function is self explanatory"""
    print ( f'{4 * 3 = } {multiply ( 4, 3 ) = }')
    print ( f'{4 + 3 = } {add( 4, 3 ) = }')
    print ( f'{4 / 3 = } {divide ( 4, 3 ) = }')
    # print(f'{4 / 3 = } {divide(4, 0)}')

    print ( (multiply - __doc__ ) )
    return None

if __name__ == '__main__':
    import doctest
    doctest .testmod ( )
    main()
    sys .exit ( 0)
```

# Unit Testing

Unit testing your code is an essential part of the software development life cycle. As code is written or modified, it must be tested before deployment and existing code must be retested (regression testing) to ensure that the software still behaves as expected (i.e. no unwanted side effects).

Most programmers have done some form of testing, with exploratory testing the most natural. Exploring the code to find bugs, issues, invalid inputs but without a plan. Manual testing is more formal, as it incorporates a list of all the features to test, different inputs and expected results. And every time a change is made, we have to go manually go through the list again. And the fun ends!

Automated testing is more thorough and dependable with far better results. In this case, the plan of tests is scripted. Python comes with a set of tools and libraries to help create automated tests. Testing is a huge topic in programming and Python but we will learn to create and execute a basic test using the Python Standard Library **unittest** module.

**Unit testing**, as the name implies is about a smaller test that checks a single component. If you are testing multiple components then that is **integration testing**. In unit testing, when we are calling a single function with parameters, this is called a test step. And when we are checking the results, this is called a test assertion. Python already has an **assert()** function.

```
>>> assert sum([10, 20, 50]) == 80, "Should be 80" # Returns an AssertionError and "Should be 80" if fails.  
>>> assert sum([1, 1, 1]) == 3, "Should be 3" # Returns an AssertionError and "Should be 80" if fails.
```

This can then be incorporated in function code:

```
from demo_calculator import add, multiply  
def test_sum():  
    assert add(10, 20) == 30, "Should be 30"  
  
def test_multiply():  
    assert multiply(10, 20) == 200, "Should be 200"  
  
if __name__ == "__main__":  
    test_sum()  
    test_multiply()
```

This is ok for simple checks, but for multiple failures we need a test runner which is a special app designed for running tests and validating output etc. The **unittest** module has both a **test framework** and **runner**. The initial difference is we put our tests into classes as methods (which we introduce in the next session) and use special assertion methods from the module rather than the built-in assert function.

```
#!/bin/python  
# Name: demo_unittest_calc.py  
# Author: QA2.0, Donald Cameron  
# Revision: v1.0  
# Description: This program will demonstrate another example of  
# creating functions with DocStrings, parameter passing, return values.  
  
#  
# Calculator program with Add, Divide and Multiply functionality.  
  
import unittest  
from demo_calculator import add, multiply, divide  
  
class TestCalculator ( unittest.TestCase ):  
    def test_add ( self ):  
        self .assertEqual ( 7, add( 4, 3 ))  
  
    def test_multiply ( self ):  
        self .assertEqual ( 12, multiply( 4, 3 ))  
  
    def test_divide ( self ):  
        self .assertEqual ( 1.333, divide( 4, 3 ))  
  
if __name__ == "__main__":  
    unittest.main()
```



# Quiz

## Packages, Modules and Namespaces Knowledge Check

5-10 mins

1. Name the command used to load a Python module?

To be revealed

2. Fill in the blank.

Python modules have a suffix?

3. How does the Python compiler know when to recompile a Python Standard Library module?

- a. Module Name
- b. Timestamps
- c. File size
- d. Importing using: from math import sin

4. Given a module called `banking.py` which has a function called `deposit()`.

Choose from the following, the code which would work:

- a. `from banking.py import deposit  
deposit(100)`
- b. `import deposit from banking  
banking.deposit(200)`
- c. `import banking  
banking.deposit(300)`
- d. `from banking import deposit  
banking.deposit(400)`

5. Name the file which is created when a new Package is created?

To be revealed

6. True/False? Does this statement satisfy PEP8 requirements?

```
>>> import sys, math, banking
```

To be revealed

# Exercises

30 mins

## Modules & Packages - exercises for 'brave' knights

In this exercise, we are going to pull together some of the scripts and functions that you have already written and load them in as modules to a top-level menu program.

Create a new script 'C:\labs\menu.py' that imports the appropriate functions from earlier lab exercises.

The script should display a menu like the following diagram, be contained within an infinite loop, and have a mechanism to quit out of the menu and the script. You should only import the functions that are required, which means you may have to modify the modules so that they have appropriate separate functions (and not solely a main() function).



1

## Modules & Packages - stretch exercises for Kings

Sadly even Kings, when looking for the holy Grail, can be arrested by the police. So we are going to really stretch you in this exercise. Write a new module called 'c:\labs\police\_data.py'. Create two functions, once called `getPoliceForces()` which returns a list of Police Forces in England, and the other `getCrimeByLoc()` which returns a list of Crimes by location (given as latitude/longitude).

This data can be accessed from "<https://data.police.uk/api/>" by sending **HTTP GET** requests. The Python standard library `requests` module can be used to send the **HTTP GET** requests.

Some sample code for each function is below and <https://data.police.uk/docs/> for help. Don't worry if you run out of time, this could be one for the evening! Extra Kudos for exception handling. [Hint: the FUN demos might help here]

```
base_url = "https://data.police.uk/api/"
```

```
full_url = base_url + "forces"
force_data = requests.get(full_url)
```

```
full_url = base_url + "crimes-at-location?lat=" + lat + "&lng=" + lon + "&date=" + date
crime_data = requests.get(full_url)
```



2

Move your piece down as you complete the exercises



**Not  
confident**

**Quite  
confident**

**Very  
confident**

Back to  
the quest





# O O P Object Oriented Programming

 Benefits of OOP

 Defining a Class

 Getter/Setter methods

 Encapsulation

 Inheritance

 Decorators

 Instantiation

 Duck Typing

 Properties

 Polymorphism

Have you met  
OOP before?

 *Add a  
Sticky Note*

# OOP

**Object-oriented programming (OOP)** is a computer programming model of structuring a program by bundling related properties and behaviours into individual objects.

In traditional or procedural programming, if we wanted to write a game with tanks, jeeps and trucks (as well as buildings, trees and lakes) we would have to describe everything as a separate variables and each thing would have to have a collection of functions describing what each thing can do. So tank1 would have a variable (data) and lots of functions (tank1\_accel, tank1\_decel, tank1\_rotate\_left etc). And we would have to replicate this for all tanks (and all other things). And we end up with a rather large source code file/s and lots of replicated code.

**OOP** recognises that in all walks of life (and programming) there is a lot of replication, for example, bank accounts, chess boards, online games etc. So in OOP, we create one blueprint called a **class** for each thing, such as a bank account, a pawn or a tank. And the class acts as a template for creating similar **objects**, but these objects are created at runtime in memory (when they are needed) and the source code file remains smaller and simpler.

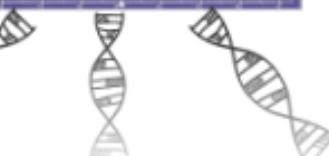
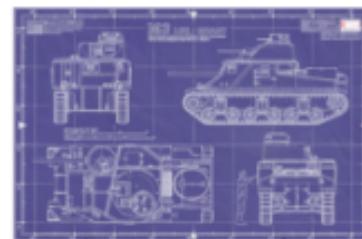
You have already seen that Python has built-in classes such as str, int, list, dict and set. Each of these has attributes (data) and things it can do to itself (methods). The type() function tells you what type an object is and the dir() function displays a directory of the attributes and method names. And in your program you can create as many instances of str objects as you want based on the str class.

Python provides tools for creating our classes (of anything) with data and methods. And we can instantiate as many objects of this class as we need. In our demonstrations, we are going to bypass bank accounts and chess pieces and create Tank objects!

Tank class	
country	_direction
model	_shells
_speed	_health
_location	
	__init__()
	accel()
	decel()
	rotate_left()
	rotate_right()
	shoot()
	take_damage()

**class Tank**

blueprint



Tank objects

# Defining classes

A few tips when creating a class:

- A class is declared using class, and membership is by indentation
- "class names use the CapWords convention" – PEP008
- Methods are declared as functions within that class, the first argument passed is the object.
- The constructor, called when a new object is created is called `_init_`
- The destructor is called `_del_` but it's rarely required and unreliable.
- Classes are usually declared in a file with the same name as the class, with .py appended
- A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition.

```
#! /usr/bin/python
# Name: tank.py
# Author: Q42.6, Donald Cawman
# Revision: v1.0
# Description: This module describes a class of Tank for
# an online game.
# 

Derived class of Task

class Task:
    __game_version = "1.0.42" # Global class variable.

    def __init__(self, country, model):
        self.country = country
        self.model = model
        self.speed = 0
        self.direction = 0
        self.location = [ 'X' : 0, 'Y' : 0, 'Y' : 0]
        self.shells = 20
        self.health = 100
        # implicitly called so we do not return.

    def accel(self, increase):
        self.speed += increase
        return None

    def decel(self, decrease):
        self.speed -= decrease
        return None

    def move_left(self, degrees):
        self.direction -= degrees % 360
        return None

    def move_right(self, degrees):
        self.direction += degrees % 360

    def shoot(self):
        self.shells -= 1
        return None

    def take_damage(self, amount):
        self.health -= amount
        return None

0000 EXTRA CODE for SPECIAL METHODS 0000

# Example of Operator overloading.
def __add__(self, other):
    return self.__health + other.__health

# Example of Duck Typing, Quack like a or object.
def __str__(self):
    return ("Health: " + self.__health + ", speed: " + self.__speed + ", shells: " + self.__shells + ")"

# Example of a GETTER method.
def get_health(self):
    return self.__health

# Example of a SETTER method.
def set_health(self, newhealth):
    self.__health = newhealth
    return None

0000 EXTRA CODE for special PROPERTIES: 0000

# Property function is wrapping the two methods with our name/interface!
tank_health = property(get_health, set_health)

0000 EXTRA CODE for DECORATORS: 0000

# Wrapping the identical Named Methods with a DECORATOR to indicate when
# they should be used. DECORATORS = PYTHONIC!
@property
def tank_health(self):
    return self.__health

# Example of a SETTER method.
@tank_health.setter
def tank_health(self, newhealth):
    self.__health = newhealth
    return None

@classmethod
def get_version(cls):
    return ("Game version: " + cls.__game_version )
```

# Instantiation

Once the class has been created (usually in its own module file). We then need to import the class and create **instances** of the class - in this case, we need to 'spawn' new tank **objects** in memory.

Once an instance of an object has been created, that object **inherits** the attributes (data) and behaviour (methods) of the class its based on. If any of the attributes or methods have an underscore prefix in their name, it identifies them as private and local to the class/object and should not be accessed outside the object. This is called **encapsulation** (hidden in a capsule). Using a double underscore prefix mangles the identifier by prefixing the name with `_modulename`.

Attributes and methods are accessed in the same way we access methods for built-in Python objects, using the dot notation, for example `str.upper()`.

Another important features of OOP is **polymorphism** (many forms) that allows us to perform a single action in different ways. For example, we could have Tank, Jeep and Helicopter objects which all have a `shoot()` method - but each shoots in a different way!

We can use the built-in functions `type()` and `dir()` to get information on a Tank object and `isinstance()` can check whether an object belongs to a class. For example, `isinstance(robin_tank, Tank)`.

```
#!/bin/python
# Name: instances.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This is an ultra realistic computer game with Tanks!
"""

Game of Tanks!

"""

import sys
import tank

def main():
    """ Main game """
    # Instantiate 3 new Tank objects.
    lancelot_tank = tank.Tank( "German" , "Tiger" )
    arthur_tank = tank.Tank( "British" , "Churchill" )
    robin_tank = tank.Tank( "American" , "Sherman" )

    # .. and the game begins.
    lancelot_tank.accel( 41 )
    arthur_tank.accel( 33 )

    robin_tank.rotate_left( 385 )
    robin_tank.accel( 15 )
    robin_tank.shoot()

    # ..and success!
    lancelot_tank.take_damage( 40 )
    arthur_tank.take_damage( 62 )

    # And now for some game visuals! Well at least a print statement!
    print ( f"Health of Lancelot's Tank = {lancelot_tank._health}" ) # Why is this POOR Code?

    return None

if __name__ == "__main__":
    main()
    sys.exit ( 0 )
```

# Special methods

In Python you will find special methods which start and end with double underscores. They are called **Magic methods** or **dunders**. They are not meant to be called directly are invoked internally from the class when a certain action occurs. For example, when you add two objects of the same class using the '+' operator, then internally the `__add__()` method will be called.

When we create a new object based on a class, the `__new__()` method is called immediately followed by the `__init__()` to initialise it. We rarely have to override `__new__()` in our class definitions, but we usually always override `__init__()` to initialise new objects.

Some more magic methods that can be overridden:

<code>__len__(self)</code>	Implement the <code>len()</code> function
<code>__str__(self)</code>	Returns a string object when the object is converted into a string.
<code>__add__</code>	+ Object can be used with 'add' operator.
<code>__sub__</code>	- Object can be used with 'minus' operator.
<code>__eq__</code>	<code>==</code> Object can be compared using the equality operator.
<code>__ge__</code>	<code>&gt;=</code> Object can be compared using the 'greater or equal' operator.
<code>__lt__</code>	<code>&lt;</code> Object can be compare using the 'less then' operator.

```
#!/bin/python
# Name: demo_special_methods.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This is an ultra realistic computer game with Tanks!
"""
Game of Tanks!
"""

import sys
import tank

def main():
    """ Main game """
    # Instantiate 3 new Tank objects.
    lancelot_tank = tank.Tank("German", "Tiger")
    arthur_tank = tank.Tank("British", "Churchill")
    robin_tank = tank.Tank("American", "Sherman")

    # ... and the game begins.
    lancelot_tank.accel(41)
    arthur_tank.accel(33)

    robin_tank.rotate_left(385)
    robin_tank.accel(15)
    robin_tank.shoot()

    # ...and success!
    lancelot_tank.take_damage(40)
    arthur_tank.take_damage(62)

    # And now for some game visuals! Well at least a print statement!
    print(f"Health of Lancelot's Tank = {lancelot_tank._health}") # Why is this POOR Code?

    # Example of Operator Overloading.
    print(f"Status of Lancelot's and Arthur's Tanks = {lancelot_tank + arthur_tank}")

    # Example of Duck Typing. Tank can now Quack like a str!
    print(f"Status of Lancelot's Tank: {lancelot_tank}")

    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# Getter & Setter Methods

In Object Oriented Programming we often want to keep the data belonging to an object private and manage what can access or modify our data. For example in a game you would not want other Tank objects knowing of your location and dire health as they would likely sneak up on you. We can limit what data is accessed using the dunder prefix but this is only a convention amongst fellow programmers and not enforced by the language.

The answer is special methods for **accessing** and **mutating** the data, more commonly called **Getter** and **Setter** methods. These methods control access and changes to the objects data, perhaps checking if you're a friendly tank or updating your health if you have received a health boost?

```
#!/bin/python
# Name: demo_getter_setter_methods.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This is an ultra realistic computer game with Tanks!
"""
Game of Tanks!
"""

import sys
import tank

def main():
    """ Main game """
    # Instantiate 3 new Tank objects.
    lancelot_tank = tank.Tank("German", "Tiger")
    arthur_tank = tank.Tank("British", "Churchill")
    robin_tank = tank.Tank("American", "Sherman")

    # .. and the game begins.
    lancelot_tank.accel(41)
    arthur_tank.accel(33)

    robin_tank.rotate_left(385)
    robin_tank.accel(15)
    robin_tank.shoot()

    # ..and success!
    lancelot_tank.take_damage(40)
    arthur_tank.take_damage(62)

    # And now for some game visuals! Well at least a print statement!
    print(f"Health of Lancelot's Tank = {lancelot_tank._health}") # Why is this POOR Code?

    # Example of Operator Overloading.
    print(f"Status of LanceLOT's and Arthur's Tanks = {lancelot_tank + arthur_tank}")

    # Example of Duck Typing, Tank can now Quack like a str!
    print(f"Status of Lancelot's Tank: {lancelot_tank}")

    # Lancelot has received a health boost!
    # Update and get health directly from private objects! Ugly and Poor!
    lancelot_tank._health = 90
    print(f"Lancelot's new health is {lancelot_tank._health}")

    # Examples of special GETTER and SETTER methods.
    # Internal methods accessing private data = better!
    lancelot_tank.set_health(100)
    print(f"Lancelot's new health is {lancelot_tank.get_health()}")

    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# Property function

Getter and Setter methods are good for managing access to private data within an object, but you likely have to use multiple different names for them and parameters are passed within parentheses.

As an alternative, Python has a built-in **property()** function that when applied to a method, it makes it behave like a variable interface with access, assignment and deletion properties (and an optional documenting property). Or in OOP speak, with getter, setter and deleter actions.

For example, if we want to access the health of a tank, assign or delete a value to the health of a tank using just one interface/variable name:

```
tank_health = property(get_health, set_health, del_health, doc_health)
```

```
#!/bin/python
# Name: demo_properties.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This is an ultra realistic computer game with Tanks!
"""
Game of Tanks!
"""

import sys
import tank

def main():
    """ Main game """
    # Instantiate 3 new Tank objects.
    lancelot_tank = tank.Tank("German", "Tiger")
    arthur_tank = tank.Tank("British", "Churchill")
    robin_tank = tank.Tank("American", "Sherman")

    # ... and the game begins.
    lancelot_tank.accel(41)
    arthur_tank.accel(33)

    robin_tank.rotate_left(385)
    robin_tank.accel(15)
    robin_tank.shoot()

    # ... and success!
    lancelot_tank.take_damage(40)
    arthur_tank.take_damage(62)

    # And now for some game visuals! Well at least a print statement!
    print(f"Health of Lancelot's Tank = {lancelot_tank._health}") # Why is this POOR Code?

    # Example of Operator Overloading
    print(f"Status of Lancelot's and Arthur's Tanks = {lancelot_tank + arthur_tank}")

    # Example of Duck Typing. Tank can now Quack like a str!
    print(f"Status of Lancelot's Tank: {lancelot_tank}")

    # Lancelot has received a health boost!
    # Update and get health directly from private objects! Ugly and Poor!
    lancelot_tank._health = 90
    print(f"Lancelot's new health is {lancelot_tank._health}")

    # Examples of special GETTER and SETTER methods.
    # Internal methods accessing private data = better!
    lancelot_tank._set_health(100)
    print(f"Lancelot's new health is {lancelot_tank._get_health()}")

    # Using a property interface with one name.
    lancelot_tank._tank_health = 101
    print(f"Lancelot's new health is {lancelot_tank._tank_health}")

return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# Decorators

A Python **decorator** is a **function** that takes another function and extends the behaviour of the latter function without explicitly modifying it. For example, the `@property` decorator is a built-in decorator for the `property()` function and can use the following three decorators - `@property`, `@<function-name>.setter`, `@<function-name>.deleter`.

All the following three methods have the same name (overloaded) but we define when we use each one by decorating it a property function, one as a getter, one as a setter and the other as a deleter.

```
@property
def tank_health(self):
```

```
    return self._health
```

```
@tank_health.setter
```

```
def tank_health(self, newhealth):
```

```
    self._health = newhealth
```

```
    return None
```

```
@tank_health.deleter
```

```
def tank_health(self):
```

```
    del self._health
```

```
    return None
```

```
#!/bin/python
# Name: demo_properties.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: This is an ultra realistic computer game with Tanks!
```

Game of Tanks!

```
import sys
import tank

def main():
    """ Main game """
    # Instantiate 3 new Tank objects
    lancelot_tank = tank.Tank("German", "Tiger")
    arthur_tank = tank.Tank("British", "Churchill")
    robin_tank = tank.Tank("American", "Sherman")

    # .. and the game begins.
    lancelot_tank.accel(41)
    arthur_tank.accel(33)

    robin_tank.rotate_left(385)
    robin_tank.accel(15)
    robin_tank.shoot()

    # .. and success!
    lancelot_tank.take_damage(40)
    arthur_tank.take_damage(62)

    # And now for some game visuals! Well at least a print statement!
    print(f"Health of Lancelot's Tank = {lancelot_tank._health}") # Why is this POOR Code?

    # Example of Operator Overloading
    print(f"Status of Lancelot's and Arthur's Tanks = {lancelot_tank + arthur_tank}")

    # Example of Duck Typing. Tank can now Quack like a sit!
    print(f"Status of Lancelot's Tank: {lancelot_tank}")

    # Lancelot has received a health boost!
    # Update and get health directly from private objects! Ugly and Poor!
    lancelot_tank._health = 90
    print(f" Lancelot's new health is {lancelot_tank._health}")

    # Examples of special GETTER and SETTER methods.
    # Internal methods accessing private data = better!
    lancelot_tank.set_health(100)
    print(f" Lancelot's new health is {lancelot_tank.get_health()})

    # Using a property interface with one name
    lancelot_tank._tank_health = 101
    print(f" Lancelot's new health is {lancelot_tank.tank_health}")

    # And testing our class method decorator.
    print(tank.Tank.get_version())
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

# Inheritance

Have you noticed that some objects in the world have got similar attributes and behaviours? For example, jeeps and trucks do similar things as tanks such as accelerate, decelerate, and turn left and right. Current accounts have got similarities to deposit and savings accounts. In chess pawns, rooks, knights have all got similar properties like colour, position on board, and have similar behaviours such as move forward or move back.

This could lead to replication of data and methods and you would be 'a very naughty boy' in the OO world if you did this. So OOP supports the concept of Inheritance, and indeed multi-inheritance, where an object can be **Derived** from one or more **Base** or **Parent** classes, and will inherit the data and methods from the Base class.

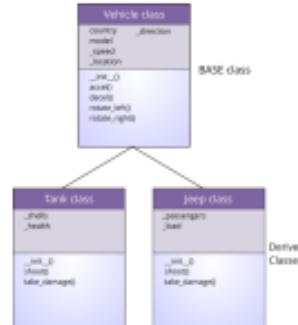
The derived or child class has all the methods and properties of the parent class, but you can add or change them.

For example, we could have a `Vehicle` base class, and `Tanks`, `Jeeps` and `Trucks` could all inherit from this one parent - thus reducing the replication of code and effort.

You might also find a class that is almost what you need, but you can't change it. If you want a string class that's slightly different from Python's str class, you can't actually change Python yourself, but could inherit from str and add new methods.

In Python, it is very common to derive our own classes from Python classes

A useful method for classes is `issubclass()` which checks whether a class is a subclass of another class.



```
#!/bin/python
# Name: vehicle.py
# Author: QIA2 0, Donald Cameron
# Revision: v1.0
# Documentation: This module is designed to interface with the vehicle
```

```
class Vehicle:
    def __init__(self, country, model):
        self.country = country
        self.model = model
        self.speed = 0

    def accel(self, increase):
        self.speed += increase
        return None

    def decel(self, decrease):
        self.speed -= decrease
```

```
    if (is_prime(num)) {  
        for (i = 1; i < num; i++) {  
            if (i * i > num) {  
                break;  
            }  
            if (i * i == num) {  
                return i;  
            }  
        }  
    }  
    return -1;  
}  
  
int main() {  
    int num = 16;  
    int result = find_factor(num);  
    cout << "Factor of " << num << " is " << result << endl;  
    return 0;  
}
```



# Quiz

OOP, Classes and Objects Knowledge Check

⌚ 5-10 mins

1. Fill in the blank.

In OOP, a class is a ? for an object.

In a Python class definition, functions are called?

- a. functions
- b. methods
- c. accessors
- d. attributes

3. Which is the correct way to instantiate a new knight based on this class?

```
class Knight:  
def __init__(self, colour, bravery):  
    self.name = f"{colour} knight"  
    self.bravery = bravery
```

- a. Knight()
- b. Knight("Green", 4, "pepperami")
- c. Knight("Black", 10)
- d. Knight.\_\_init\_\_("Green", 4)

4. True or False - a class can inherit from multiple base classes?

To be revealed

# Exercises

⌚ 35 mins

1

## OOP - exercises for stupidly brave Black Knights

In this exercise you are going to open an existing module called 'c:\labs\movieDB.py' which will contain one simple class called MovieDB. It will have the following attributes and methods:



We have provided some of the code already for creating the class, opening and closing the DB connection and created a new table if required. The DB is a SQLite3 database called "C:\labs\movie\_db\_sqlite".

You have to create new methods for inserting new rows, delete existing rows, query all rows and commit all changes.

[Hint: you will find the table definition in an existing method]

```
# Author: http://www.jasonhurley.com
# Author: Q4.2 LIVE, V1.0
# Description: This module defines a class called MovieDB for creating objects
# that can interact with a SQLite3 database file.
# ...
MovieDB class.
"""
import sqlite3

class MovieDB:
    # Class variable storing location of DB
    DB_LOC = "C:\labs\movie_db_sqlite"

    def __init__(self):
        """
        :param:
        self._db_conn = sqlite3.connect(MovieDB.DB_LOC)
        self.cur = self._db_conn.cursor()
        except Exception as err:
            raise ValueError(err)
        # No return as not explicitly called.

    def create_table(self, table_name="movies"):
        self.cur.execute("""CREATE TABLE IF NOT EXISTS movies
        (id INTEGER PRIMARY KEY
        ,title VARCHAR(100)
        ,year VARCHAR(100)
        ,rating INTEGER)
        """)
        return None

    def commit(self):
        """
        Commit changes to database
        :param:
        self._db_conn.commit()
        return None

    def __del__(self):
        """
        Close connection automatically when object is deleted
        self._db_conn.close()
        return None
```

Move your piece down as you complete the exercises



2

Write a simple program to import and test the MovieDB class.

**Not  
confident**

**Quite  
confident**

**Very  
confident**

Back to  
the quest



# Solutions

⌚ 5 mins

## Quiz

1. Name three ordered and two unordered collections?

Ordered?

Str,  
tuple,  
list

Unordered

dict,  
set

2. Is a tuple mutable?

True

False

3. The joy of sets! What would be printed from the following code?

```
scottish_movies = {'Brigadoon', 'Local Hero', 'Scot土her', 'The Last of the Mohicans', 'The Joy of Sets', 'The Windup Girl'}
```

{'Local Hero'}

4. Fill in the blanks:

Dictionaries have unique

Keys

Sets have unique

Objects

5. What would be printed from the following code?

```
scottish_movies = {'Brigadoon', 'Local Hero', 'Scot土her', 'The Last of the Mohicans', 'The Joy of Sets', 'The Windup Girl'}
```

['Local Hero', 'Comfort and Joy']

6. Write the print statement from question 5 using set methods?

```
print(scottish_movies.intersection(fav_movies))
```

```
#!/bin/python
# Name: lottery.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: Generates 6 Random Unique Lottery numbers.
#---

# Lottery number generator. Good luck and remember your names!
#---

import sys
import random

def main():
    """ The Main Program """
    lots = set() # Did we mention we want Unique numbers!
    while len(lots) < 6:
        num = random.randint(1, 50)
        lots.add(num)

    print ("Lottery numbers = ", lots)
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

1

```
#!/bin/python
# Name: topTen.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: Display top movies.
#---

# Top movies and Rankings.
#---

import sys

def main():
    movies = []

    file_in = open(r'C:\Users\dk\Downloads\top250_movies.txt', 'r', mode='rt')
    movies = file_in.readlines() # Read movies into list.
    file_in.close()

    for rank, movie in enumerate(movies, start=1):
        print(f'{rank} {movie}', end='')

    print(f'\nFirst movie = {movies[0]}')
    print(f'\nLast movie = {movies[-1]}')

    topN = int(input("Choose Top-N movies: "))
    for rank, movie in enumerate(movies, start=1):
        print(f'{rank} {movie}', end=' ')
        if rank == topN: break

    print(f'\n{topN} top movies = {movies[0:topN]}')

if __name__ == "__main__":
    main()
    sys.exit(0)
```

2

```
#!/bin/python
# Name: topTen_stretch.py
# Author: QA2.0, Donald Cameron
# Revision: v1.0
# Description: Display top movies.
#---

# Top movies and Rankings.
#---

import sys

def get_movies():
    file_in = open(r'C:\Users\dk\Downloads\top250_movies.txt', 'r', mode='rt')
    movies = file_in.readlines()
    file_in.close()
    return movies

def display_movies(movies, topN):
    for rank, movie in enumerate(movies, start=1):
        print(f'{rank} {movie}', end=' ')
        if rank == topN: break
    return None

def search_movies(text, movies):
    for rank, movie in enumerate(movies, start=1):
        if text in movie:
            print(f'{rank} {movie}', end=' ')
    return None

def main():
    movies = []
    while True:
        menu = """
        Menu Options
        -----
        1. Get online movie ranking from IMDB
        2. Display top ranking movies
        3. Search for movie
        -----
        """
        print(menu)
        option = input("Enter option: ")
        if option == "1":
            movies = get_movies()
        elif option == "2":
            topN = int(input("Choose topN movies: "))
            display_movies(movies, topN)
        elif option == "3":
            text = input("Enter movie to search: ")
            search_movies(text, movies)
        return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```

3-7

# Solutions



? Quiz

1. What is the default mode for the built-in `open()` function?

"rt" [Read and Text mode]

4. Which Python standard library module preserves ONE Python object to a file?

:pickle.py

- ## 2. What is the default buffering mode for text files?

### Line buffering

5. Which Python standard library module preserves MULTIPLE Python objects to a file?

shelve.py

- ### 3. Which methods are used for random access?

`.seek()` and `.tell()`

1-4

5

```
#!/usr/bin/python
# Name: display_movies.py
# Author: Q/A/02, Donald Canham
# Revision: v1.0
# Description: Display movie information
# IMDB movie information
# IMDB
# import sys

def main():
    # Solution for Quiz 2 & 3
    fIn = open('C:\Users\Q/A/02\op250_movies.txt', 'r', encoding='utf-8')
    for movie in fIn:
        print('Title: ' + movie.title() + ', and ' + movie.year)
    fIn.close()

    # Solution for Quiz 4
    fIn = open('C:\Users\Q/A/02\op250_greatest.txt', 'r', encoding='utf-8')
    for rank, movie in enumerate(fIn, start=1):
        print('Rank: ' + rank + ' (movie title: ' + movie.title() + ', and ' + movie.year + ')')
    fIn.close()

    return None

if __name__ == '__main__':
    main()
    sys.exit(0)
```

```

    System
    Name movieLibrary
    Author Q426. Donald Cameron
    Version v1.0
    Description Movie database movie information file

    Persistent Movie Information
    Author Q426

    import sys

    def main():
        movies = [
            {"Title": "The Shawshank Redemption", "Director": "Frank Darabont", "Year": 1994},
            {"Title": "The Godfather", "Director": "Francis Ford Coppola", "Year": 1972},
            {"Title": "The Godfather: Part II", "Director": "Francis Ford Coppola", "Year": 1974},
            {"Title": "The Dark Knight", "Director": "Christopher Nolan", "Year": 2008},
            {"Title": "The Godfather", "Director": "Francis Ford Coppola", "Year": 1971}
        ]

        file_out = open("C:/Users/110101099/movies.txt", "w", encoding="utf-8")

        for movie in movies:
            print(f'{movie["Title"]},{movie["Year"]},{movie["Director"]}', file=file_out)

        file_out.close()
        return None

    if __name__ == "__main__":
        main()

```

```
 1 Assignment  
 2   - Name: msvr_dbsrc.py  
 3 Author: QAL2.0, Donald Cannon  
 4 Revision: v1.0  
 5 Description: Powershell: Monitor dictionary on file  
 6  
 7 Preserving PowerShell information
```

```

    print("running memory")
    th_out = None
}

def open_compressed_file():
    th_in = gzip.open("test.pck", "rb")
    th_out = pickle.Unpickler(th_in)
    th_in.close()
    return th_out

```

```
6 #!/bin/python
# Name: display_unused_ports.py
# Author: QAA2.0, Donald Cameron
# Revision: v1.0
# Description: Display all the unused ports in the SERVICES
#
```

```

Display command next ports from the SERVICES.SLIC file
[...]
import sys

def main():
    if sys.platform == "win32":
        filename = r"C:\WINDOWS\System32\drivers\etc\services"
    else:
        filename = r"/etc/services"

    # Open file handle for reading in text mode
    file_in = open(filename, mode="rt")
    all_ports = set(range(1, 201))
    used_ports = set()

    # Iterate through file handle and read the services
    for line in file_in:
        if line.isspace() or continue:
            if line.startswith("#") or continue:
                fields = line.split(" ")
                if fields[0].split(" ")[0] == "port":
                    port = int(fields[1][0] + fields[1][1])
                    if port >= 200:
                        used_ports.add(port)

    print("All ports = %s" % all_ports)
    print("Used ports = %s" % used_ports)
    print("Unused ports = %s" % (all_ports - used_ports))

    file_in.close()

if __name__ == "__main__":
    main()

```

# Solutions



1. What does B.R.E and E.R.E stand for

Basic and Extended Regular Expressions

2. What prefix should you use with your Regex patterns?

"pattern" for Raw pattern strings:

3. Can you think of a better way to write the following pattern, which is 1 char longer?

卷之三

[15] \$

4. Given this code, what will the pattern match

one took a "Brave like Robin ran away, Braveley ran away away. Run away  
run away his ugly ugly day, he bravely turned his tail and fled."  
one in a [run-away](#)(r"{})(domain).r(1); bank, [\[tagname:329808CAE\]](#)  
one [println](#)(n-group[1])

Brave Sir Robin ran away. Bravely ran away away. When danger reared its ugly head, he brave

5. Given this code, what would be printed?

```
    row back = "backslash"
    row as = row.asarray() + "%".join(["-"]).decode("utf-8")
    row print(as.groupby([]), as.groupby([])[0], as.groupby([])[1])
```

(r, 'a', 'c')

```
#! bin/python
#
# Name: top250.py
# Author: QIAO, Donald Cameron
# Revision: v1.0
#
# Description: Download the top 250 movies chosen by IMDb users.
#
# Download and display online movie information.
#
# Import modules
#
import sys
import imdb
import re
#
def main():
    ia = imdb.IMDb()
    pattern = input("Enter movie search string: ")
    for rank, movie in enumerate(ia.get_top250_movies(), start=1):
        if re.search(pattern, str(movie), re.IGNORECASE):
            print("%d %s" % (rank, movie))
    return None
if __name__ == "__main__":
    main()
    sys.exit(0)
```

5-6

# Solutions

5 mins

## Quiz

1. What does (\*) at the beginning of the parameter list do?

Enforce named parameter passing.

3. True or False. Do Python functions always return a value?

True  
(Python functions always return None unless explicit value)

2. What is the output of the following function call?

```
def my_func(num):  
    return num + 10  
my_func(10)  
print(num)
```

NameError: name 'num' is not defined

4. What is the output of the following function call?

```
def display_info(*args):  
    for a in args:  
        print(a)  
display_info('breakfast', 'years'1982')
```

movie  
year

#!/bin/python  
# Name: calc.py  
# Author: QA2.0, Donald Cameron  
# Revision: v1.0  
# Description: A Calculator program

Calculator program with add, divide, multiply, subtract and modulus functionality.  
\*\*\*  
import sys  
  
def add (x, z):  
 """ Return sum of x and z """  
 return x + z  
  
def divide (x, z):  
 """ Return x divided by z to 3 decimal places """  
 return x / z  
  
def multiply (x, z):  
 """ Return product of x and z """  
 return x \* z  
  
def subtract (x, z):  
 """ Return difference of x and z """  
 return x - z  
  
def modulus (x, z):  
 """ Return remainder of x divided by z """  
 return x % z

# Solution for Question 7

```
# def add(*numbers):  
#     """ Return sum of all parameters """  
#     return sum(numbers)  
  
# def multiply(*numbers):  
#     """ Return the product of all parameters """  
#     product = 1  
#     for num in numbers:  
#         product *= num  
#     return product  
  
def main():  
    """ This is the main program function """  
    print ("11 + 5 = " + add(11, 5))  
    print ("11 / 5 = " + divide (11, 5))  
    print ("11 * 5 = " + multiply (11, 5))  
    print ("11 - 5 = " + subtract (11, 5))  
    print ("11 % 5 = " + modulus (11, 5))  
  
    if print("11 + 5 = [add(11, 5, 33, 15, 6.5)]")  
    if print("11 * 5 = [multiply(11, 5, 33, 15, 6.5)]")  
    return None
```

```
if __name__ == "__main__":  
    main()  
    sys.exit (0)
```

1-3

#!/bin/python  
# Name: search250.py  
# Author: QA2.0, Donald Cameron  
# Revision: v1.0  
# Description: Download the top 250 movies chosen by IMDB users.  
\*\*\*  
# Download and display online movie information  
\*\*\*  
import sys  
import math  
import re  
  
def search\_movie ( pattern ):  
 """ Search for a Regex pattern in the top 250 movies """  
 is = math.IMDB()  
 count = 0  
 for rank, movie in enumerate ( is.get\_top250\_movies(), start = 1):  
 n = is . search ( pattern , str ( movie ), re . IGNORECASE ):  
 if n:  
 print ("[" + rank + "] " + movie + ")")  
 count += 1  
 print ("Matched: " + str ( count ) + " lines")  
 return None  
  
def main ():  
 """ The Main Program """  
 search\_movie ("\*the\*") # 50 lines  
 search\_movie ("\*0-9\*") # 11 lines  
 search\_movie ("\* [a-z ] [0-9]\*") # 5 lines  
 search\_movie ("\*([a-z ])\*[0-9]\*") # 15 lines  
 search\_movie ("\*([a-zA-Z ])\*\$") # 1 line  
 search\_movie ("\*movie\*") # 1 lines  
 search\_movie ("\*([a-zA-Z ])\*([a-zA-Z ])\*") # 8 lines  
 return None  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()  
 sys.exit (0)

4

#!/bin/python  
# Name: searchWords.py  
# Author: QA2.0, Donald Cameron  
# Revision: v1.0  
# Description: Search and display lines in multiple files using  
# Regular expressions  
\*\*\*  
# Patterns searching tool  
\*\*\*  
import re  
import \_io  
  
def search\_pattern ( pattern , \* files ):  
 """ Search for Regex patterns in multiple files """  
 if not files : files = ("A/C/1/absurd.txt")  
 for file in files:  
 fh = open ( file , mode = "rt" )  
 for line in fh :  
 n = re . search ( pattern , line )  
 if n:  
 print ( line , end = "" )  
  
 fh . close ()  
  
 return None  
  
def main ():  
 """ Main Program """  
 search\_pattern ("([A-Z ])\*[0-9]\*")  
 search\_pattern ("([A-Z ])\*[0-9]\*", "C/absurd.txt", "C/absurd2.txt")  
 return None  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()  
 sys.exit (0)

5

# Solutions

5 mins

## Quiz

1. Can you think of a better way to write the following pattern, which is 16 chars long?

```
300-[num for num in range(0, 10) if num % 2 == 0 and num % 4 == 0]
```

```
[0, 4, 8] # List Comprehension
```

2. Can you think of a better way to write the following pattern, which is 16 chars long?

```
300-guysby = ['eric', 'michael', 'mervy', 'john', 'terry', 'graham']  
300-[name for name in guysby if len(name) <= 5.]
```

```
['eric', 'terry', 'john'] # 3 Displayed
```

3. Can you think of a better way to write the following pattern, which is 15 chars long?

```
300-movies = ['Casablanca', '2021', 'Mulan', '2020', 'Mary Poppins Returns', '2018']  
300-film = movies.copy()  
300-movies[0] = 1998  
300-film == movies
```

```
False # Shallow Copy. Movies was changed after the copy.
```

1

```
#! /bin/python  
# Name: gen1.py  
# Author: QA2.0, Donald Cameron  
# Revision: v1.0  
# Description: Floating point version of the built-in range()  
# function.  
  
# Generate a sequence of floating point numbers  
# (step=0.1, stop=0.25)  
  
import sys  
  
def frange (start, stop, step=0.25):  
    """ Generate a sequence of floating point numbers  
    if step == 0, return []  
  
    cur = float (start)  
    while cur < stop:  
        yield cur  
        cur += step  
  
def main():  
    """ The Main Program """  
    print (list (frange (1.1, 1.7)))  
    print (list (frange (1, 3, 0.01)))  
    print (list (frange (1, 3, 100)) # Should return [1.0, 2.0] not [1, 2])  
    print (list (frange (1, 100))) # Should return an empty list  
    print (list (frange (1, 3, 0.00))) # Should return an empty list  
    print (list (frange (-1,- 0.5, 0.1)))  
    print (frange (1,2)) # Should print <generator object frange at 0x.  
  
    for num in frange (3.142, 1.2):  
        print ("{:12.2f}\n".format (num))  
    return None  
  
if __name__ == "__main__":  
    main()  
    sys.exit (0)
```

2

```
#! /bin/python  
# Name: gen2.py  
# Author: QA2.0, Donald Cameron  
# Revision: v1.0  
# Description: Floating point version of the built-in range()  
# function. Improved version that can accept ONE parameter.  
  
# Generate a sequence of floating point numbers.  
# (range(len), stop, [step=0.25])  
  
import sys  
  
def frange (start, stop=None, step=0.25):  
    """ Generate a sequence of floating point numbers  
    if step == 0, return []  
    if step == None:  
        step = start  
        start = 0.0  
    else:  
        start = float (start)  
  
    while start < stop:  
        yield start  
        start += step  
  
def main ():  
    """ The Main Program """  
    result1 = list (frange (0, 3.5, 0.25))  
    result2 = list (frange (3.5, 0))  
  
    if result1 == result2:  
        print ("Default worked!")  
    else:  
        print ("Oops! [result1] is not equal to [result2]!")  
  
    return None  
  
if __name__ == "__main__":  
    main()  
    sys.exit (0)
```

3

```
#! /bin/python  
# Name: gen3.py  
# Author: QA2.0, Donald Cameron  
# Revision: v1.0  
# Description: Floating point version of the built-in range()  
# function. Improved version using Decimal module.  
  
# Generate a sequence of accurate floating point numbers.  
# (range([start], stop, [step=0.25]))  
  
import sys  
import decimal  
  
def frange (start, stop=None, step=0.25):  
    """ Generate a sequence of floating point numbers  
    if step == 0: return []  
    step = decimal.Decimal (str (step))  
  
    if step == None:  
        step = decimal.Decimal (str (start))  
        cur = decimal.Decimal (0)  
    else:  
        step = decimal.Decimal (str (step))  
        cur = decimal.Decimal (str (start))  
  
    while cur < stop:  
        yield float (cur)  
        cur += step  
  
def main ():  
    """ The Main Program """  
    print (list (frange (1.1, 3.0)))  
    print (list (frange (1, 3, 0.33)))  
    print (list (frange (1, 3, 100)) # Should return [1.0, 2.0], not [1, 2])  
    print (list (frange (3, 1))) # Should return an empty list  
    print (list (frange (1, 3, 0))) # Should return an empty list  
    print (list (frange (-1,- 0.5, 0.1)))  
    print (frange (1,2)) # Should print <generator object frange at 0x.  
  
    for num in frange (3.142, 1.2):  
        print ("{:12.2f}\n".format (num))  
    return None  
  
if __name__ == "__main__":  
    main()  
    sys.exit (0)
```

# Solutions

5 mins

## Quiz

1. Name the command used to load a Python module?

import

2. Fill in the blank.

Python modules have a `.py` suffix?

3. How does the Python compiler know when to recompile a Python Standard Library module?

- a. Module Name
- b. Timestamps
- c. File size
- d. Importing using: `from math import sin`

4. Given a module called `banking.py` which has a function called `deposit()`.

Choose from the following, the code which would work:

- a. `from banking.py import deposit`  
`deposit(100)`
- b. `import deposit from banking`  
`banking.deposit(200)`
- c. `import banking`  
`banking.deposit(300)`
- d. `from banking import deposit`  
`banking.deposit(400)`

5. Name the file which is created when a new Package is created?

`__init__.py`

6. True/False? Does this statement satisfy PEP8 requirements?

`>>> import sys, math, banking`

- True
- False

# PEP8 recommends import on separate lines

```
if __name__ == "__main__":
    main()

# Name: menu.py
# Author: QAZB0, Donald Cesar
# Revision: v1.0
# Description: Top level menu program.
# ...
# Menu of reusable functions.
# ...
import sys
from searchWords import search_pattern # Option 1.
from display_movies import display_top250 # Option 2.
from search250 import search_movie # Option 3.
from display_unused_ports import unused_ports # Option 4.

def main():
    """The Main Program"""
    menu = """
    Menu Options
    -----
    1. Search for pattern in file
    2. Display top250 movies
    3. Search top250 movies
    4. Display unused ports
    ...
    """

    while True:
        print (menu)
        option = input ("Enter option (1-4, q=quit) ")
        if option == "1":
            pattern = input ("Enter search pattern: ")
            search_pattern (pattern)
        elif option == "2":
            display_top250()
        elif option == "3":
            pattern = input ("Enter search pattern: ")
            search_movie (pattern)
        elif option == "4":
            unused_ports()
        elif option.lower () == "q":
            break
        else:
            print ("Invalid option")
    return None

if __name__ == "__main__":
    main()
    sys.exit (0)
```

```
if __name__ == "__main__":
    main()

# Name: police_data.py
# Author: QAZB0, Donald Cesar
# Revision: v1.0
# Description: Get online Police Data
# ...
# Get and display Online Police Crime Data
# ...
import sys
import requests

def getPoliceData():
    """Get List of Online Police Crime Data"""
    base_url = "https://data.police.uk/api/"
    full_url = base_url + "forces"
    force_data = requests.get (full_url).json ()[0]
    print (force_data)

def main():
    """The Main Program"""
    menu = """
    Police Data
    -----
    1. List of Police Forces
    2. Crime by location
    ...
    """

    while True:
        print (menu)
        option = input ("Enter option (1,2, q=quit) ")
        if option == "1":
            getPoliceData ()
        elif option == "2":
            getCrimeByLoc ()
        elif option.lower () == "q":
            break
        else:
            print ("Invalid option")
    return None
```

# Solutions



5 mins

## Quiz

### 1. Fill in the blank.

In OOP, a class is a **blueprint/template** for an object.

### 2. In a Python class definition, functions are called?

- a. functions
- b. methods
- c. accessors
- d. attributes

### 3. Which is the correct way to instantiate a new knight based on this class?

```
class Knight():
    def __init__(self, colour, bravery):
        self.name = f'{colour} knight'
        self.bravery = bravery
```

- a. Knight()
- b. Knight("Green", 4, "pepperami")
- c. Knight("Black", 10)
- d. Knight.\_\_init\_\_("Green", 4)

### 4. True or False - can a class inherit from multiple base classes?

True  False

```
#!/usr/bin/python
# Name: movieDB.py
# Author: QAZ0. Donald Cannon
# Revision: v1.0
# Description: This module defines a class called MovieDB for creating objects
# that can interact with a SQL(Lite) database file.

# MovieDB class
# ...
# import sqlite3
# ...
# No names as not explicitly called

class MovieDB:
    # Class variable holding location of DB
    DB_LOC = "C:\lab\movie_db.sqlite"

    def __init__(self):
        try:
            self._db_conn = sqlite3.connect(MovieDB.DB_LOC)
            self._cur = self._db_conn.cursor()
        except Exception as err:
            raise ValueError
        # No names as not explicitly called

    def create_table(self, table_name="movies"):
        self._cur.execute(f'''CREATE TABLE IF NOT EXISTS {table_name}
        (id INTEGER PRIMARY KEY
        , title VARCHAR(30)
        , year VARCHAR(10)
        , rating INTEGER)''')
        return None

    def insert_row(self, id, title, year, rating, table_name="movies"):
        """ Insert new row into table """
        sql = f"INSERT INTO {table_name} VALUES ({id}, {title}, {year}, {rating})"
        self._cur.execute(sql)
        return None

    def delete_row(self, row_id, table_name="movies"):
        """ Delete row from db table """
        sql = f"DELETE FROM {table_name} WHERE id={row_id}"
        self._cur.execute(sql)
        return None

    def query_all_rows(self, table_name="movies"):
        """ Delete row from db table """
        sql = f"SELECT * FROM {table_name}"
        self._cur.execute(sql)

        rows = self._cur.fetchall()
        for row in rows:
            print(row)
        return None

    def commit(self):
        """ Commit changes to database """
        self._db_conn.commit()
        return None

    def __del__(self):
        """ Close connection automatically when object is deleted """
        self._db_conn.close()
        return None
```

```
#!/usr/bin/python
# Name: test_movieDB.py
# Author: QAZ0. Donald Cannon
# Revision: v1.0
# Description: Instantiate and test our MovieDB objects
# ...
# State and Retrieve movie data to db using a MovieDB object
# ...
# import sys
# import movieDB

def main():
    """ The Main Program """
    movies = movieDB.MovieDB() # Instantiate MovieDB object
    movies.create_table()
    movies.insert_row(1, 'Brave', '2012', 10)
    movies.insert_row(2, 'Braveheart', '1995', 10)
    movies.insert_row(3, 'Babe', '1995', 10)
    movies.commit()

    movies.query_all_rows()
    movies.delete_row(1)
    movies.delete_row(2)
    movies.delete_row(3)
    movies.commit()
    return None

if __name__ == "__main__":
    main()
    sys.exit(0)
```



# PYTHON IN PRACTICE

Apply Activity:

**Workplace Application Plan**



Any questions?

- Review all questions in the Apply Phase documentation
- Ensure you have completed all of the relevant elements
- If you haven't done so already, discuss your plan with your line manager
- Complete the Apply work
- Reflect on your experience
- Once you have completed all the required steps confirm this has been done and have this validated by your line manager
- Finally, submit your work
- We will then check your submission, mark your activity as complete and issue you with your **Python in Practice** badge.

**Good luck!**