

# Python Labs

Contents

[Python Labs 1](#_Toc607102)

[Introduction 2](#_Toc607103)

[Beginner Exercises 3](#_Toc607104)

[**Hello World!** 3](#_Toc607105)

[**Assignment** 3](#_Toc607106)

[**Parameters** 3](#_Toc607107)

[**Parameters/Operators** 3](#_Toc607108)

[**Conditionals** 3](#_Toc607109)

[**Conditionals 2** 3](#_Toc607110)

[**Recursion** 3](#_Toc607111)

[**Recursion/Lists** 3](#_Toc607112)

[**Recursion/Lists** 4](#_Toc607113)

[**User input** 4](#_Toc607114)

[**Functions** 4](#_Toc607115)

[Intermediate Exercises 4](#_Toc607116)

[**Blackjack** 4](#_Toc607117)

[**Unique Sum** 4](#_Toc607118)

[**Too hot?** 5](#_Toc607119)

[**Leap Year** 5](#_Toc607120)

[**Paint Wizard** 5](#_Toc607121)

[Advanced 6](#_Toc607122)

[**Prime Numbers 1** 6](#_Toc607123)

[**Prime Numbers 2** 6](#_Toc607124)

[**Strings** 6](#_Toc607125)

[**Strings 2** 6](#_Toc607126)

[**Battleships** 6](#_Toc607127)

# Introduction

This document contains a range of exercises in order to help you get to grips with Python. You are not expected to complete *all* tasks, but attempting most of the exercises will help to cement your Python knowledge.

For all exercises, try to make use of debugging to help you complete the solutions.

For some exercises you complete, try to also create files that log the date and time the code runs (so, every time the code runs, a line should be written to a file saying “Program name ran at time on date”).

# Beginner Exercises

## **Hello World!**

Output “Hello World!” to the console.

## **Assignment**

Store “Hello World!” in a variable, then output it to the console.

## **Parameters**

Create a method that gets a string from the user, and then outputs that string to the console.

## **Parameters/Operators**

Create a method that accepts two integers as an input, then returns an integer that is a sum of the two integers given.

## **Conditionals**

Modify your method from the Parameters/Operators task to accept another parameter, a Boolean, which if it is true, the method will print a sum of the two numbers, and if it is false it will print the multiplication of the two numbers.

## **Conditionals 2**

Modify your method from the Conditionals task to have another if statement that checks if one of the numbers is 0, if this is true then print the other non-0 number.

**Input -> 1, 0 Return 1**

**Input -> 1, 2 Return 3**

## **Recursion**

Create a **for** loop that will call and output the result of your method from the

Conditionals 2 task 10 times.

**Lists**

Create a list with 10 integer values in it, then call and output the first element in the list.

## **Recursion/Lists**

Using your list that you created in the Lists task, create a loop that iterates through your list, outputting the values contained within it.

## **Recursion/Lists**

Create a loop that populates a list with values, outputting them at each iteration. Then create another loop that iterates through the array, changing the values at each point to equal itself times 10, outputting them at each iteration.

***Example Output***

1,2,3,4…

10,20,30,40…

## **User input**

Modify the previous task to take input from the user, taking an integer off of the user and using that integer to determine how large the array is going to be.

## **Functions**

Create a function that asks the user for a number and whether they want to double or triple the number. Have methods within the function for doubling and tripling the user’s number.

# Intermediate Exercises

## **Blackjack**

Given 2 integer values greater than 0, return whichever value is closest to 21 without going over 21. If they both go over 21 then return 0

**Input(18,21) -> 21**

**Input(20,18) -> 20**

**Input(22,22) -> 0**

## **Unique Sum**

Given 3 integer values, return their sum. If one value is the same as another value, they do not count towards the sum. Aka only return the sum of unique numbers given. **Input(1,2,3) -> 6**

**Input (3,3,3) -> 0**

**Input (1,1,2) -> 2**

## **Too hot?**

Given an integer value and a Boolean value, **temperature** and **isSummer**, if temperature falls between 60 and 90 (inclusive), unless its summer where the upper limit is 100 instead of 90, return True. If temperature falls outside this range, return false.

## **Leap Year**

Given a year work out if it is a leap year or not.

**Rule:** A year is a leap year if it is divisible by 4, and either divisible by 400 or not divisible by 100.

## **Paint Wizard**

Create a paint requirement wizard that will calculate which of the following three products, would be the cheapest to buy, for the room you are painting.

Work out which one wastes the least.

Work out which is the best choice for any room (Cheapest).

1. **CheapoMax (20Litre) £19.99**

**This tin of paint will cover 10m2 per Litre**

1. **AverageJoes (15 Litre) £17.99**

**This tin of paint will cover 11m2 per Litre**

1. **DuluxourousPaints (10 Litre) £25**

**This tin of paint will cover 20m2 per Litre**

# Advanced

## **Prime Numbers 1**

Create an algorithm that determines how many prime numbers are between 1 and 3 million.

**Extension –** Have it finish running in under 2 minutes

## **Prime Numbers 2**

Create an algorithm that determines how many prime numbers are between 1 and 2 **billion**.

**Extension** – Have it finish running in under 3 minutes

## **Strings**

Given two strings, write a program that efficiently finds the longest common subsequence.

## **Strings 2**

Given two strings, write a program that outputs the shortest sequence of character insertions and deletions that turn one string into the other.

## **Battleships**

Create the battleships game!

This project is to create a digital version of the popular board game known as battleships. Battleships is a two player game with 2 phases. In the first phase the player’s ships are placed on the board. In the second phase each player takes it in turns to select grid squares on the board in an attempt to find and destroy their opponent’s ships. Once one player has lost all of their ships the game is over and the player who still has ships on the board is the winner.

Each player has a number of ships including: 2 patrol boats (1 x 2), 2 battleships (1 x 3), 1 submarine (1 x 3), 1 destroyer (1 x 4) and 1 carrier (1 x 5).

There are a number of rules that players must follow.

* 2 Ships cannot occupy the same space on the board.
* If a player scores a ‘hit’ on their opponent, then they get a second shot.
* Ships cannot be moved once they have been placed.

**Advice**

Battleships is a seemingly simple strategy game but without careful planning it can be easy to become “lost” in the project. It is recommended that you attempt to complete the project in a set of stages where with each stage you increase the level of complexity. Remember that as the complexity of your project increases you may find that you wish to go back to a previous version in some cases so it is highly advised that you create versioned copies of your project at each stage. An advised set of stages are:

Stage 1:

A 3 x 3 grid with one ship that is 2 pieces long is placed on in the grid.

Stage 2:

A 3 x 3 grid with 2 ships that are 2 pieces long and placed on the grid with validation to ensure legal placement.

Stage 3:

Two 3 x 3 grids with 2 ships where players take alternating turns taking shots at the other grid.

Stage 4:

Differentiation between ‘hits’ and ‘misses’ implemented.

Stage 5:

Checks for sunk ships with game over when one player has lost all their ships.

Stage 6:

Two 12 x 12 grids with all 7 ships placed in valid locations.

Stage 7:

Players can select the placement of their ships on the grid during phase 1.

Stage 8:

Each turn logged to a file

Stage 9:

Implementation of an AI player.

Stage 10:

Players can only see their own grid with shots taken.

Stage 11:

Implementation of a GUI has been attempted.

**ATM Machine**

Create

* New file: **06ATMAdvanced.py**

Code a program that:

* Simulates the behaviour of an Automatic Teller Machine (ATM) or Cash Machine

Save and run:

* Check the contents of the file using Notepad
* Use a debugging tool to debug

*Brief*

Northern Frock needs you to write a program for their new ATMs (or Automatic Teller Machines). In this version, you will need to use a customer database that will initially look like this (ignore the top row):

ID Title First Name Surname Balance

1057 Mr. Jeremy Clarkson 172.16

2736 Miss Suzanne Perry 15.62

4659 Mrs. Vicki Butler-Henderson 23.91

5691 Mr. Jason Plato 62.17

Write a program that will do the following:

* Generate the above data file.
* Prompt the user for their ID number, validating against the list of known users.
* If the ID number 99999 (5x 9s) is given the machine should shut down.
* Print a menu:

Welcome to Northern Frock

* 1 - Display balance
* 2 - Withdraw funds
* 3 - Deposit funds
* 9 - Return card
* Enter an option:
* If ‘1’ is entered, display the current balance and the maximum amount available for withdrawal (must be a multiple of £10)
* If ‘2’ is entered, provide a sub-menu with withdrawal amounts:
  + *Please select withdrawal amount (1-£10, 2-£20, 3-£30, 4-£60, 5-£80, 6-£100, 7-Other Amount, 8-Return to Main Menu)*
  + If ‘1 to 6’’ is selected check that the requested withdrawal is allowed, print a message to show that the money has been withdrawn, calculate the new balance and return to main menu.
  + If ‘7’ is selected, then prompt the user for an integer value. Check this number is a multiple of 10 and that the withdrawal is permitted, print a message to show that the money has been withdrawn, calculate the new balance and return to main menu.
  + If ‘8’ is selected return to main menu.
* If ‘3’ is entered, provide another menu that will allow the user to enter an amount to deposit (does not need to be a multiple of £10), return to main menu or return card. If funds are deposited, provide appropriate feedback and update the balance.
* If ‘9’ is entered, print a goodbye message and return to the initial prompt (for the next customer).
* If another value is entered, print an error message and print the menu again.

Extensions:

* Add an extra field to the database for a 4-digit PIN which should be prompted for and checked following the entry of the ID number. The user should also have the option to change their PIN.
* Add another field to record whether the card is blocked. Three incorrect PIN attempts should permanently lock the card. PIN attempts should only be reset by correctly entering the PIN. Simply removing the card and trying again should not work.
* Create an encoding algorithm that will store the PIN in a format that cannot be read by just looking at the file. Only by running the PIN attempt through the same algorithm can the match be found.