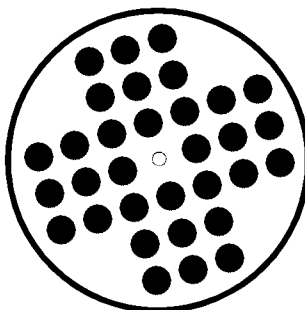
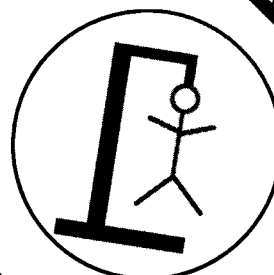
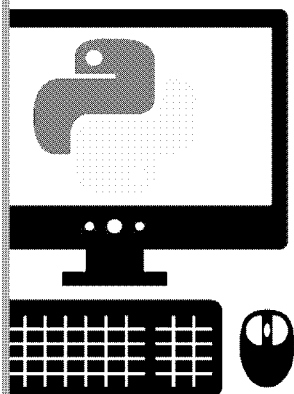




**Computer Science**

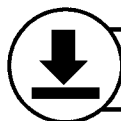
GCSE (9–1) | AQA | 8520



**2016 specification**  
First assessment from 2018

# Python Exercises

*for AQA GCSE (9–1) Computer Science*



Download support files from  
[zzed.uk/productsupport](http://zzed.uk/productsupport)

[zigzageducation.co.uk](http://zigzageducation.co.uk)

**POD**  
**9834**

Publish your own work... Write to a brief...  
Register at [publishmenow.co.uk](http://publishmenow.co.uk)

# Contents

<b>Thank You for Choosing ZigZag Education.....</b>	<b>ii</b>
<b>Teacher Feedback Opportunity .....</b>	<b>iii</b>
<b>Terms and Conditions of Use.....</b>	<b>iv</b>
<b>Teacher's Introduction .....</b>	<b>1</b>
<b>AQA (8520) Specification Mapping .....</b>	<b>2</b>
<b>Exercises .....</b>	<b>4</b>
Exercise 1 – Number Game.....	4
Exercise 2 – Rock, Paper, Scissors.....	6
Exercise 3 – Turtle Drawing.....	8
Exercise 4 – The Monty Hall Problem .....	10
Exercise 5 – Caesar Cipher.....	12
Exercise 6 – Check Digits .....	15
Exercise 7 – Hangman.....	18
Exercise 8 – Peg Solitaire .....	20
Exercise 9 – Blackjack Hands .....	23
Exercise 10 – Connect Four.....	26
<b>Answers.....</b>	<b>29</b>
Exercise 1 – Number Game.....	29
Exercise 2 – Rock, Paper, Scissors.....	32
Exercise 3 – Turtle Drawing.....	34
Exercise 4 – The Monty Hall Problem .....	36
Exercise 5 – Caesar Cipher.....	38
Exercise 6 – Check Digits .....	40
Exercise 7 – Hangman.....	42
Exercise 8 – Peg Solitaire .....	44
Exercise 9 – Blackjack Hands .....	48
Exercise 10 – Connect Four.....	52
<b>Extras.....</b>	<b>56</b>
Python Quick Help Sheet.....	56
Python Common Error Guide.....	59

# Teacher's Introduction

This resource is designed to support the development of programming skills using Python. It contains 10 unique exercises, featuring a range of scenarios that develop the core programming principles.

These include using arrays, iteration, selection, sequence, string manipulation, arithmetic and Boolean operators and file handling – all requirements of the AQA GCSE (9–1) Computer Science specification (a specification map is provided on pages 2-3, showing how the relevant specification content is covered across the exercises).

Each exercise contains a combinations of questions and tasks, and consists of two sections – A and B.



The purpose of **Section A** is to test knowledge of the existing code, and to fix any errors that might be present. The skill of debugging is incredibly important in programming as programmers rarely tend to write whole programs by themselves.

**Section B** provides students with the opportunity to develop the functionality; these should take slightly longer to complete and will help students when they start addressing the NEA element of the course.

Along with the worksheets, there are Python<sup>v3.5</sup> programs that should be changed as the questions have been answered. Working Python files are provided for every worksheet, along with written answers.

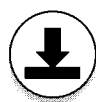
*Note that credit should also be given for any valid responses that are not explicitly included in this resource.*

The following icons are used to indicate the nature of the task, along with the number of marks available.

	A written response is required (using the answer lines provided)
	An amendment to the electronic skeleton program is required

In addition, the following additional resources are provided to assist students:

- *Python Quick Help Sheet* – provides an overview of the most commonly used Python (3.5) commands, along with examples. It is recommended that students also refer to the official Python documentation at: <https://docs.python.org/>
- *Python Common Error Guide* – describes and gives examples of a number of common pitfalls, along with the corrected code. A useful reference for students, particularly when attempting to debug their code.



The following resources are provided as a download via the ZigZag Education Support Files system, which can be accessed via [zzed.uk/productsupport](https://zzed.uk/productsupport)

- Skeleton Python script that students need to modify for each exercise
- Exemplar Python script (with all of modifications made) for each exercise

## Free updates

Register your email address to receive any future free updates\* made to this resource or other Computer Science resources your school has purchased, and details of any promotions for your subject.

Go to [zzed.uk/freeupdates](https://zzed.uk/freeupdates)

\* resulting from minor specification changes, suggestions from teachers and peer reviews, or occasional errors reported by customers

# AQA (8521) SPECIFICATION M

	Exercise 1 Rock, Paper, Scissors	Exercise 2 Turtle Drawing	Exercise 3 Monty Hall Problem	Exercise 4 Caesar Cipher	Exercise 5 Ch...
<b>2.1 – Data types</b>					
Understand the concept of data types	✓		✓		
Understand the following data types: integer, real, Boolean, character, string	✓		✓		
<b>2.2 – Programming concepts</b>					
Use and understand: variable/constant declaration, assignment, iteration, selection, subroutines	✓	✓	✓	✓	
Use definite and indefinite iteration	✓	✓	✓	✓	
Use nested selection and iteration		✓	✓		
<b>2.3 – Arithmetic operations in a programming language</b>					
Be able to use: addition, subtraction, multiplication, real division, integer division (including remainders)	✓		✓	✓	
<b>2.4 – Relational operations in a programming language</b>					
Be able to use: equal to, not equal to, less than, greater than, less than or equal to, greater than or equal to	✓		✓		
<b>2.5 – Boolean operations in a programming language</b>					
Be able to use: NOT, AND, OR	✓	✓			

INSPECTION COPY

COPYRIGHT  
PROTECTED



	<b>Exercise 1</b>	<b>Exercise 2</b>	<b>Exercise 3</b>	<b>Exercise 4</b>	<b>Exercise 5</b>	<b>Exercise 6</b>
	<b>Number Game</b>	<b>Rock, Paper, Scissors</b>	<b>Turtle Drawing</b>	<b>Monty Hall Problem</b>	<b>Caesar Cipher</b>	<b>Check Digits</b>
<b>2.6 – Data structures</b>						
Understand the concept of data structures					✓	✓
Use arrays in the design of solutions to simple problems				✓	✓	✓
<b>2.7 – Input/output and file handling</b>						
Obtain user input from the keyboard		✓		✓	✓	✓
Output data and information from a program to the display	✓	✓		✓	✓	✓
Read/write from/to a text file					✓	
<b>2.8 – String handling operations in a programming language</b>						
Be able to use: length, position, substring, concatenation, convert character to character code, convert character code to character, string conversion operations	✓				✓	✓
<b>2.9 – Random number generation in a programming language</b>						
Be able to use random number generation				✓		
<b>2.10 – Subroutines (procedures and functions)</b>						
Understand the concept of subroutines	✓	✓	✓	✓	✓	✓
Use subroutines that return values	✓				✓	✓
Use local variables	✓	✓		✓	✓	✓
<b>2.12 – Robust and secure programming</b>						
Be able to write simple validation routines	✓	✓				✓

INSPECTION COPY

COPYRIGHT  
PROTECTED



# EXERCISE 1 – NUMBER GAME

This is a simple game whereby one person (the computer in this case) thinks of a number between 1 and 100. The other person then has to guess what the number is.

If they guess incorrectly they are given clues about whether their guess is too high or low and they have to guess again until they get it right. The idea is to guess the number correctly in as few guesses as possible.

A program designed to play the game is shown below and provided electronically).

Study the code and try to understand what is happening in the program, before attempting the exercises.

```

1  import random
2
3  def guess():
4      num = input("Please enter your guess")
5      return num
6
7  print("Welcome to the number guessing game")
8  print("The objective is to guess the number I'm thinking of")
9  print("I will give you clues after your first guess.")
10 secretNumber = random.randint(1,100)
11 print("I have thought of a number from 1-100")
12 numGuessed=guess()
13 if numGuessed < secretNumber:
14     print("Guess is too low, guess higher!")
15 else:
16     print("Guess is too high, guess lower!")

```

## SECTION A

A 1

The program does not run properly and you should get a syntax error. Identify the cause of the problem and fix the program.

Program updated ☐

A 2

When the program asks the user to enter their guess, it is not formatted properly. Modify the program so that it presents a more suitable layout/prompt.

Program updated ☐

A 3

The welcome message does not stand out – it is merged into the request for the first guess and the instructions.

Modify the welcome message so that it appears underlined to separate it from the text, and then leave a blank line after the instructions before the user enters their first guess.

Program updated ☐

A 4

Currently, the program will generate a type error when you run it – this is because the number that is being entered is actually stored as a string. On computers, the user is not aware of writing such as "23" to integers such as 23. This is so that the computer can treat it as a number. For example, "23" + "23" is actually "2323" on a computer, but 23 + 23 is 46. This is why the computer needs to know whether it is a string or an integer.

Find the error in your program and fix it.

Program updated ☐

**COPYRIGHT  
PROTECTED**



**A 5**

The program only allows the user to have one guess before stopping.

Modify the program that it keeps asking the user to enter a guess until

*Hint: You will need to use a new variable that is initialised before the ne*

Program updated ☐

**A 6**

The program will not perform correctly when the user guesses the num

Investigate what happens and describe it, then fix the program electro



Program updated ☐

## SECTION B

**B 1**

Develop the program further so that the game prints out the number  
the user took to get the correct answer.

Program updated ☐

**B 2**

It is important to add validation to programs to prevent errors from  
user input.

Modify the program to:

- only allow the user to enter numbers from 1–100
- print out an error message when they enter an invalid number
- ask them to try again.



Program updated ☐

**B 3**

Currently, if a user enters anything other than an integer, the program

Fix this issue in your program using a TRY...EXCEPT.

Remember that the guess function should not exit/return until a va

Program updated ☐



**COPYRIGHT  
PROTECTED**



## EXERCISE 2 – ROCK, PAPER, SCISSORS

Rock, paper, scissors is a game played by two people in order to decide on the outcome something, much like tossing a coin. Both players tap their fist three times and then make either a rock (clenched fist), paper (open, flat hand) or scissors (two fingers open like scissors and the others clenched). The following rules are then used to decide who wins

- Rock beats scissors (because it smashes them).
- Paper beats rock (because it wraps it).
- Scissors beats paper (because scissors cut paper).
- If both players show the same hand then it's a draw.

A program designed to play the game is shown below (and provided electronically). Study the code carefully to understand what is happening in the program, before attempting to run it.

```

1  def printRules()
2      print("The computer will think of either rock, paper or scissors")
3      print("You will enter r for rock, p for paper or s for scissors")
4      print("The computer will reveal it's choice and then you choose")
5      print()
6
7  def playGame():
8      choice=input("Enter r for rock, p for paper or s for scissors")
9      computerChoice=random.randint(0,2) # 0=rock, 1=paper, 2=scissors
10
11     if computerChoice == 0:
12         print("The computer chose: Rock")
13     elif computerChoice == 1:
14         print("The computer chose: Paper")
15     else:
16         print("The computer chose: Scissors")
17
18     if choice == r:
19         if computerChoice == 0:
20             print("It's a draw")
21         elif computerChoice == 1:
22             print("Computer Wins!")
23         else:
24             print("Player Wins!")
25
26     print("Welcome to the Rock, Paper, Scissors Game")
27     print("=====")
28     printRules()
29     playGame()
  
```

There are a number of syntax errors in the code, which will need to be fixed before it can be run.

A **syntax error** means that we have not followed the rules of the programming language. The error that we have given is slightly wrong (e.g. a missing bracket or quotation mark).

### SECTION A

A 1

The first error is on line 1 which defines the function printRules(). Identify the issue and fix the program accordingly.

Program updated ☐

A 2

There is a second syntax error within the printRules() function. Identify the issue and fix the program accordingly.

Program updated ☐

**COPYRIGHT  
PROTECTED**





A 3

In Python, we use = to assign a value to a variable but we use == to compare the value of one variable to another (or against a specified value).

Find a place where an incorrect number of = has been used and fix the program accordingly. State the line number below.

Program updated ☐

A 4

There is another syntax error on line 14. Identify the issue and fix the program accordingly.

Program updated ☐

A 5

There is a problem on line 18. Identify the issue and fix the program accordingly.

Program updated ☐

A 6

There is another syntax error on line 20. Identify the issue and fix the program accordingly.

Program updated ☐

A 7

There is one final syntax error on line 28. Identify the issue and fix the program accordingly.

Program updated ☐

A 8

The program is now giving a NameError, random not defined. Describe the issue below and fix the program accordingly.

Program updated ☐

A 9

There is now a logic error in the program. Test it by playing the game using the options to see what happens. Identify the issue and fix the program accordingly.

Program updated ☐

## SECTION B

B 1

The player can currently enter something other than r, p or s. If the program should be robust and ask them to re-enter their choice. Develop the program implementing this validation rule.

Program updated ☐

B 2

The user should be allowed to enter an uppercase R, P or S, not just lowercase. Develop the program further to implement this additional function.

Program updated ☐

B 3

The program currently only plays the game once. Modify the program to allow the user if they would like to play again instead of just pressing enter to exit.

Program updated ☐

## EXERCISE 3 – TURTLE DRAWING

One of the first robots invented for drawing was the turtle – a simple robot with a choice of pens that drives around the floor and either has a pen touching the paper underneath it (using different colours), or has the pen raised so that it can move to a new location to start the next part of the drawing (or the next drawing).

To create turtle drawings, Python's turtle package is used. This package is called 'turtle' as it allows you to control a 'virtual turtle' that draws wherever the turtle has moved to.

This is an example of an **external library**. Some basic commands have been given below.

Command	Description
<code>turtle.forward(x)</code>	Move x pixels in the direction the turtle is pointing.
<code>turtle.left(x)</code>	Turns the turtle left x degrees.
<code>turtle.right(x)</code>	Turns the turtle right x degrees.
<code>turtle.color(x)</code>	Sets the colour using a value represented using hexadecimal.
<code>Turtle.penup()</code>	Puts the pen up which means that the turtle can be moved without creating a line.
<code>Turtle.pendown()</code>	Puts the pen down which means that the turtle will draw a line when it moves.
<code>turtle.heading()</code>	Returns the direction in which the turtle is heading (in degrees). If the turtle is heading east, it's heading is 0.
<code>turtle.setheading(x)</code>	Sets the direction in which the turtle is facing.
<code>turtle.speed(x)</code>	Allows the speed of the turtle to be changed (1 = slowest, 10 = fastest).

A basic program that draws a square has been given below.

Study the code and try to understand what is happening in the program, before attempting to write your own.

```

1  import turtle
2
3  BLUE="#0000ff"
4  PINK="#ff00ff"
5  GREEN="#00ff00"
6
7  def drawSquare(size, colour):
8      turtle.color(colour)
9      for i in range(4):
10         turtle.forward(size)
11         turtle.right(90)
12
13  turtle.speed(5)
14  turtle.setheading(0)
15  turtle.pendown()
16  drawSquare(100, BLUE)
17  turtle.penup()
18  turtle.forward(5)
19  turtle.right(90)
20  turtle.forward(5)
21  drawSquare(80, PINK)
22
23  turtle.exitonclick()
```

**COPYRIGHT  
PROTECTED**



## SECTION A

A 1

The program draws a blue square and then the turtle attempts to draw a pink square. No square appears. Identify the issue and fix the program so that the pink square appears.

Program updated ☐

A 2

After the bug described above has been fixed, you notice that the pink square is in the wrong place – it's not inside the blue square.

By looking at the position of the turtle, identify the issue and fix the program.



Program updated ☐

A 3

Modify the program to make the colour of the inside square green instead of pink. You should only change one line.

Program updated ☐

A 4

The colour codes on lines 3–5 are in hexadecimal. Modify the program to use a new constant storing the hex colour for RED (look it up if you need to). Change the colour of the outside square to RED.

Program updated ☐

A 5

The position of the inside square is not even. Modify the program so that the inside square is positioned evenly inside the outside square.

Program updated ☐

## SECTION B

B 1



Not all users can be expected to know how every function of every module works. They need to be able to look up the documentation to fill any gaps.

By looking up the documentation for the turtle module, modify the program to use the `hideturtle()` command that will make turtle invisible after the image has been drawn. Add the command used below.

<https://docs.python.org/3.5/library/turtle.html>

Program updated ☐

B 2

Other shapes can be drawn easily using the turtle. What isn't quite so easy is drawing a shape without taking the pen off the paper, or drawing the same line twice. One example that can be achieved is shown on the right.

Create a function called `drawHouse()` that draws this shape using turtle.

You can start at any point you like. You will need to use Pythagoras' theorem to calculate the lengths of the lines to draw – a calculator is recommended for the main house.

Program updated ☐

B 3



Create another draw function to replicate the star shown on the right. Use a loop to create the shape. Attempt to colour in the star using the `fill()` command.

Program updated ☐

COPYRIGHT  
PROTECTED

## EXERCISE 4 – THE MONTY HALL PROBLEM

Consider the following scenario:

You are on a TV game show, and have the choice of three doors to open. One of the doors has a brand-new car behind it – the other two have old goats behind them.

Once you have picked a door, the game show host opens one of the two doors that you did not pick, to show you a goat. He then offers you a choice to switch your choice to the remaining door.

Should you switch your choice? Does it make any difference to how likely you are to win? If you do not switch you have a 1/3 chance of winning, and if you do switch, the odds double and you have a 2/3 chance of winning.

You will be using a simulation of the Monty Hall Problem. A basic program is shown electronically.

Study the code and try to understand what is happening in the program, before attempting the questions.

```
1  door = ["goat", "goat", "car"]
2
3  choice = input("Door 1, 2 or 3? ")
4  otherDoor = 0
5  goatDoor = 0
6
7  if choice == 1:
8      if door[1] == "goat":
9          otherDoor = 3
10         goatDoor = 2
11     elif door[2] == "goat":
12         otherDoor = 2
13         goatDoor = 3
14 elif choice == 2:
15     if door[0] == "goat":
16         otherDoor = 3
17         goatDoor = 1
18     elif door[2] == "goat":
19         otherDoor = 1
20         goatDoor = 3
21 elif choice == 3:
22     if door[0] == "goat":
23         otherDoor = 2
24         goatDoor = 1
25     elif door[1] == "goat":
26         otherDoor = 1
27         goatDoor = 2
28
29  switch = input("There is a goat behind door " + goatDoor + ". Do you want to switch to door " + otherDoor + "? (y/n) ")
30
31
32  if switch == "y":
33      choice = otherDoor
34
35  if door[choice-1] == "car":
36      print("You won a car!")
37  else:
38      print("You lost a goat!")
```

### SECTION 1

A 1

Describe the purpose of the '\n' symbol used on line 29.

INSPECTION COPY

COPYRIGHT  
PROTECTED



**A 2**

After making a choice, the program crashes unexpectedly.  
Describe the reason for this, and fix the program accordingly.

Program updated ☐

**A 3**

The program does not require a user's choice, even if it is valid.  
Describe the reason for this, and fix the program accordingly.



Program updated ☐

**A 4**

Currently, the prize is always behind the same door.  
Fix this electronically by using `random.shuffle()`.

Program updated ☐

**A 5**

On line 35, the selection statement refers to **choice-1**.  
Why does it refer to this instead of simply **choice**?

## SECTION B

These tasks are to enable you to build a simulation or computer model for the Monty Hall problem. The program will run automatically with a random choice for you to intervene in each situation. It will also set different variables to allow you to see how many times to play the game or whether you wish to see that you can win the car. The program will automatically run the model for thousands of games and see what happens.

**B 1**

To observe the effect of switching doors, it would be useful to be able to play multiple games. Modify the program so that 10 games are played before the program ends. Name the type of programming construct used to achieve this below.

Program updated ☐

**B 2**

In a simulation, you should not need to manually enter your choice. Introduce randomness to pick for us.  
Change the program so that instead of asking for an input, it automatically chooses a door.

Program updated ☐

**B 3**

Create a subroutine `monty_hall` using the code that you have written. The subroutine should take in two parameters – the first should indicate the door that should be chosen, and the second should indicate whether you should switch doors or not.



Run the program 1,000 times with switching, and 1,000 times without switching. Record the number of times that you won the car, and how many times you lost.

Program updated ☐

**COPYRIGHT  
PROTECTED**



## EXERCISE 5 – CAESAR CIPHER

Life in Ancient Rome was very different to how we live in the present day. In 44 BC, there were no computers, no cars, and if you wanted to send a message to your friend who lived in another city, you either had to deliver it yourself or pay someone to deliver it for you.

The problem that Julius Caesar had, during the sending of messages, was that military messages were often being stolen or read during delivery. To combat this issue, he devised one of the very first forms of **encryption**. In this method, each letter is shifted along by a fixed amount to turn the message into ciphertext. If the secret key is 2, then A → C, ..., Z → B.

**Encryption:** The act of scrambling a message in a way that only the intended recipient can read it.

A program that performs a basic Caesar cipher is shown below (and is provided as a file for you to download). Study the code and try to understand what is happening in the program, before attempting the questions.

```

1  def letter_to_number(letter):
2      letters = "abcdefghijklmnopqrstuvwxyz"
3      # Find the number corresponding to the given letter
4      # In this case a = 0, b = 1, c = 2, ..., z = 25.
5      number = letters.index(letter)
6      return number
7
8  def number_to_letter(index):
9      letters = "abcdefghijklmnopqrstuvwxyz"
10     # Finds the letter corresponding to the given number
11     # In this case 0 = a, 1 = b, 2 = c, ..., 25 = z.
12     return letters[index]
13
14  def shift(letter):
15      n = letter_to_number(letter)
16      return number_to_letter((n + 13) % 26)
17
18  def rot13(string):
19      cipher = ""
20      for letter in string:
21          ciphertext += shift(letter)
22      return ciphertext
23
24
25  plaintext = "i love computing!"
26
27  ciphertext = rot13(plaintext)
28  print(ciphertext)

```

### SECTION A

A 1

What is the value that each letter has been shifted by in this program?

.....

A 2

Initially, the program does not run correctly.

Identify the type of the error (including naming the type of the error) and suggest a way to fix it accordingly.



Program updated ☐

**COPYRIGHT  
PROTECTED**



**A 3**

A table has been provided that will let you manually decode the given

0	1	2	3	4	5	6	7	8
a	b	c	d	e	f	g	h	i
13	14	15	16	17	18	19	20	21
n	o	p	q	r	s	t	u	v

The output from the program is **'w ybir pbzchgwat!'**.



Using the message using the table above, you can deduce that the message was **'hello world!'**. Write the line that it occurred on below.

Program updated ☐

**A 4**

To encrypt another message, you have to change the program's source code. Ideally, the program should ask the user to input a string to be encrypted.

State below how you would ask the user for input in Python. Update your program electronically to reflect this.

Program updated ☐

**A 5**

When a string is input that contains uppercase letters, the program fails. Modify your program electronically to allow the user to input uppercase letters without the program failing. State the function that you use.

Program updated ☐

## SECTION B

**B 1**

ROT13 is a special case of the Caesar cipher, as to decrypt a previous message you simply run the function `rot13()` on it again. If the shift is different from the one used in ROT13, to decrypt a message you need to use a different shift value (a negative amount).

Create a new function, `encrypt()`, that takes a string and a 'shift value' and returns the encrypted string.

You will need to use the function `shift()`. You will also need to use the function `ord()`.



Write the encryption of the phrase 'hello world!' with a 'shift value' of 13.

Program updated ☐

**COPYRIGHT  
PROTECTED**



**TRIVIA (Attention all Mathematicians!!)**

The Python command % is often referred to in pseudo code as MOD. MOD returns the remainder after integer division has occurred. The function `10 MOD 5` will return 0 but `11 MOD 5` will return 1.

Although the function `-9%3` returns 0 as expected, `-10%3` returns 2 whereas `10%3` returns 1. The function `-5%20` does not return -5 or 5, as you might expect. What does `-5%20` return?

Why do you think that is (mathematically)?

Hint: Computers cannot do division (or multiplication) directly, so they use repeated addition.

B	2
---	---



To decrypt a message, you need to shift the letters in the opposite direction to the encryption. Write a function `decrypt()` that takes in a string and a 'shift value', that shifts the string in the opposite direction.

Using your function, decrypt the string "drsc sc k combod wocckq" which was encrypted by shifting the letters 10 spaces forwards.

Program updated ☐

B	3
---	---

If a user needs to encrypt a lot of text, it is more sensible to read it from a file. Changing lines 25–27, load the text file "plaintext.txt" into your program using a "shift value" of 13.

Program updated ☐



**COPYRIGHT  
PROTECTED**





## EXERCISE 6 – CHECK DIGITS

At many stages of communication between hardware devices there is a chance that a message is misunderstood. Check digits are especially useful, as they can tell us if there has been a mistake.

One use of check digits is on ISBN numbers, used for book identification. Each new book has a unique 13-digit code. The thirteenth digit is reserved for a check digit that verifies that the ISBN is correct.

To calculate the check digit of a 12-digit unique book identifier, use the following arithmetic:

Split the unique identifier:	9 7 8 - 1 4 7 - 1 1 7 9 0
Multiply every other number by 2:	9 14 8 2 1 1 3 1 2 1 9 0
Add all of the numbers:	9+21+8+3+4+21+1+3+1+21+9+0 = 101
Perform division modulo 10:	101 MOD 10 = 1
Subtract the result from 10:	10 - 1 = 9 ← This is the check digit.

**NOTE:** if the sum modulo 10 = 0, the check digit is 0.

This particular ISBN is written in the following way: 978-1-47-111790-9

The first number, 978, indicates that the 13-digit code is used for identifying a book. There are different document types; for instance, the number 979 is used for sheet music, which is used for musical notation.

The second number, 1 (after the dash), indicates the book is from an 'English-speaking area'. There are different codes depending on the country – the first few have been listed below.

0	English-speaking area	2	French-speaking area	4	Japan
1	English-speaking area	3	German-speaking area	5	(former) USSR

The third and fourth numbers identify the publisher, the book, and the edition of the book. The last digit is the check digit used for error checking. An ISBN checking program has been given below (code is in Python).

```

1 def ISBNcheck(ISBN):
2
3     # Split the ISBN into the unique ID and the check digit
4     unique_id = []
5     for i in range(len(ISBN)-1):
6         unique_id.append(int(ISBN[i]))
7     actual_check_digit = ISBN[-1:]
8
9     # Multiply the second, fourth, sixth, ... elements by 3
10    times_three=[]
11    for x in range(len(unique_id)):
12        if x%2 == 0:
13            times_three.append(unique_id[x]*3)
14        else:
15            times_three.append(unique_id[x])
16
17    # Calculate the sum of the numbers
18    sum_new_digits = 0
19    for x in range(0,len(times_three)):
20        sum_new_digits += times_three[x]
21
22    # Take the sum mod 10, and subtract from 10
23    sum_mod_10 = sum_new_digits % 10
24    if sum_mod_10 == 0:
25        sum_mod_10 = 10
26    check_digit = 10 - sum_mod_10
27
28    # Check if the calculated check digit is equal to the actual check digit
29    if check_digit == actual_check_digit:
30        return "valid."
31    else:
32        return "invalid."
33
34    choice = input("Enter the ISBN number: ")
35    print("ISBNcheck() returns " + ISBNcheck(choice))

```

**COPYRIGHT  
PROTECTED**



## SECTION A

A	1
---	---

The code on line 7 takes the last character of the array and assigns it to a digit. There is a problem with the data types on this line which will mean the ISBN will return invalid.

Fix the error electronically, and write the line that it occurred on below.

Program updated ☐

A	2
---	---

Testing your program on the (valid) ISBN 9781471117909, you'll see that it does not work correctly. The error is in the iteration statement that sums the digits.

Explain below why the error is happening, and fix your program electronically.

Program updated ☐

A	3
---	---

Finding the value of `sum_new_digits` on lines 11–13 can actually be written in one line instead of three, using an inbuilt function.

Change lines 11–13 so the sum of the array is calculated in one line. State below the name of the inbuilt function that you have used.

Program updated ☐

A	4
---	---

In this program, the user can enter a 13-digit number and check that it is correct. Add length validation to your program to only continue if the user has entered a 13-digit number.

Program updated ☐

A	5
---	---

The program currently exits immediately before you can see the output. Describe how this could be prevented, and implement this change to your program.

Program updated ☐

## SECTION B

B	1
---	---

Instead of checking for a 13-digit number, the user should enter a properly formatted ISBN, e.g. 9-78-1471117909, where, excluding the dashes, there are still thirteen digits. Update your program to reflect this – a message should show if the ISBN is not correctly formatted.

Hint: The functions `.split()` and `"".join()` will be useful.

Program updated ☐

**COPYRIGHT  
PROTECTED**



**B 2**

Explain why returning "True" and "False" is better than returning "y".  
Change your program electronically to do this.

.....

.....

Program updated ☐

**B 3**

The second number in ISBN (after the first dash) denotes either a book, or the country from which the publisher originates.



If the ISBN is valid, your program should print out the country that

To do this, write a function called `getCountry()`, that converts the second number to a country name.  
The first six codes have been given below (you can assume that any other codes are for other countries).

0	English
1	English
2	French
3	German
4	Japan
5	(former) USSR

Program updated ☐



INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## EXERCISE 7 – HANGMAN

Hangman is a popular pencil and paper game where one player has to guess the word, one letter at a time. If the guessing player guesses incorrectly, a stick figure is drawn one line at a time. The game is over after a certain number of guesses.

The origins of Hangman are fairly unknown, with a guess that it was first played during the Victorian era, as hanging was one of the more popular methods of execution at the time. The game needed to be created about it is a mystery, however.

To play this game on a computer, words are treated as **strings**.

**String:** A data type that can be seen as a list of characters.

A program that contains the functions needed to play hangman is shown below (and is available on the Zig Zag Education website). Study the code and try to understand what is happening in the program, before attempting the exercises.

```

1  print("Welcome to Hangman!\n")
2
3  word = list("computing")
4  guessed_word = list("_____")
5  lives = 10
6  wordGuessed = False
7
8  while lives>=1 and not wordGuessed:
9
10     print(" ".join(guessed_word))
11
12     user_guess = input("Guess a letter/word! (" + str(lives) + " lives remaining)\n")
13
14
15     # Check if letter is in the word
16     letter_in_word = False
17     for i in range(len(word)):
18         if user_guess == word[i]:
19             guessed_word[i] = user_guess
20             letter_in_word = True
21
22     if letter_in_word == False:
23         lives -= 1
24
25     if guessed_word == word:
26         print(" ".join(guessed_word))
27         print("You have guessed the word correctly!")
28         wordGuessed=True
29     elif lives > 1:
30         print("You failed to guess the word correctly")

```

### SECTION A

A 1

Initially, when the program is run, an error immediately occurs. Explain the error below, and fix it electronically.

.....

.....



A 2

Explain the purpose of the characters "\n" and "\" in lines 12–13.

.....

.....

**COPYRIGHT  
PROTECTED**



**A 3**

The program tells you that you have lost after your first guess, this is a bug. Fix your program electronically and state the line(s) that you have changed.

.....

.....

Program updated ☐

**A 4**

Describe why a guess of 'C' does not do anything, even though it is in the word "computing". Fix your program on the computer.



Program updated ☐

**A 5**

Describe the purpose of the join() function on line 10 of the program.

.....

.....

## SECTION B

**B 1**

The user should also be able to guess the whole word, as well as the letters. Remarkably, this added functionality only requires changing one line of code.

By changing **one line** of your program, add functionality that compares the user's guess to the whole word – if the guess is correct, the game should stop.



*Hint: There is already code that does something similar to this – what can you include this extra requirement?*

Program updated ☐

**B 2**

So far, the Hangman game only uses one word – "COMPUTING". This is pretty boring – it would be much better if multiple words could be used.

In your electronic program, change lines 3 and 4 to read a random word from the provided text file, "words.txt".

*Hint: You will also need to change the way that the underscore mask is created.*

Program updated ☐

**B 3**

Finally, it would be nice if the user could see a list of all of the previous guesses they have entered.



Add an array called 'guessed\_letters' to your program, that stores the letters that are **successful**. You will also need to print out this list for the user at the end of the game.

Program updated ☐

**COPYRIGHT  
PROTECTED**



## EXERCISE 8 – PEG SOLITAIRE

Peg solitaire is a one-player game consisting of a board with 33 holes and 32 pegs. The aim is to remove pegs by jumping over them, in a similar fashion to draughts (or checkers).

For example, if in a line you had **abXc** – with X representing a space – you could move the leftmost peg into the space, removing the 'b' peg (moving **ac**).

You will be making an electronic version of the game but on a 4 × 4 board.

A basic Python peg solitaire game is shown below (and is provided electronically).

Study the code and try to understand what is happening in the program, before attempting the tasks.

```
1 def find(board, choice):
2     for i in range(1, len(board)):
3         for j in range(1, len(board[0])):
4             if choice == board[i][j]:
5                 return i, j
6
7 grid = [["a", "X", "c", "d"], ["e", "f", "g", "h"], ["i", "j", "k", "l"],
8         ["m", "n", "o", "p"]]
9 while True:
10    print(grid)
11    choice = input("Enter a letter: ")
12    direction = input("Enter a direction (u,d,l,r): ")
13
14    row, column = find(grid, choice)
15
16    if direction == "r":
17        if column + 2 < len(grid[0]):
18            if grid[row][column + 2] == "X":
19                grid[row][column] = "X"
20                grid[row][column + 1] = "X"
21                grid[row][column + 2] = choice
22
23    if direction == "l":
24        if column - 2 >= 0:
25            if grid[row][column - 2] == "X":
26                grid[row][column] = "X"
27                grid[row][column - 1] = "X"
28                grid[row][column - 2] = choice
29
30    if direction == "u":
31        if row - 2 >= 0:
32            if grid[row - 2][column] == "X":
33                grid[row][column] = "X"
34                grid[row - 1][column] = "X"
35                grid[row - 2][column] = choice
36
37    if direction == "d":
38        if row + 2 < len(grid):
39            if grid[row + 2][column] == "X":
40                grid[row][column] = "X"
41                grid[row + 1][column] = "X"
42                grid[row + 2][column] = choice
```

**NOTE:** In this program, it will not close itself unless you have completed the tasks in the program press Ctrl+C.

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## SECTION A

A 1

The 'find' function cannot find letters on the first column or the first row. Explain below why this is the case. You should also fix the problem elsewhere.

.....  
 .....  
 Program updated ☐

A 2

Instead of finding 'X' for the blank spaces, it has been decided that a space symbol will be used instead.

Describe why the current program is not particularly good if you need space symbol frequently, and how the program could be improved.

Change your program appropriately, making the consideration that it might be used frequently.

.....  
 .....  
 Program updated ☐

A 3

The program does not print the 2D array in a readable manner.

Write a function show() that takes the board as a parameter, and prints it out.

*You should change line 10 so your function is actually used.*

Program updated ☐

A 4

Add validation to your program to ensure that a choice is in the correct range. For example, entering "C" and "R" on the first turn does not move 'c' left.

Program updated ☐

A 5

Having an IF statement inside an IF statement inside an IF statement is not good programming practice.

State below the proper name that we give to 'IF inside an IF inside an IF' and change your program so this practice is not used.

.....  
 Program updated ☐

## SECTION B

B 1

When moving, the program assumes that the space next to the letter is empty.

Describe what happens, and fix your program to only make a move if the space is jumping over a different one.

.....  
 .....  
 Program updated ☐

COPYRIGHT  
PROTECTED

**B 2**

It would be useful if the player could save their progress in the game. When asked for a letter, the program should save the state of the board.

You should save the board by storing the size and then the contents of the board. Write the number of rows onto the first line of the file and the number of columns onto the second line. Then write the contents of each space on a new line in a text file called 'board.txt'.

You should also allow the user to load the board. When the user enters 'load', the program should read the number of rows and columns from the file and set the board size accordingly. Then read the contents of the board from the file and store it in the program.

Program updated ☐

**B 3**

The board can be extended to include larger sizes. Currently, the board is generated at the start of the program. Instead, it should be generated at the start of the game.

Given that the ord('a') = 97, extend your program to allow custom letters. You should set the space that would have the letter 'b' to be the empty space.

The grid size should be set in the program – the user does not need to enter it.

Program updated ☐

**B 4**

The game has been won if there is one piece remaining.

Add a check to your program that tests if the game has been won.

Test that your function works by playing on a 4 x 1 grid.

Program updated ☐



**COPYRIGHT  
PROTECTED**





## EXERCISE 9 – BLACKJACK HANDS

Blackjack is one of the more well-known card games, played all over the world by millions of people in their homes and in casinos. It is a fairly unique game, in that the dealer only has a slight edge over the player – and players can actually gain a slight advantage if they learn how to count cards (although this is frowned upon by gambling establishments!).

The given program automatically plays blackjack. It repeatedly 'hit' (get a new card) and only 'stick' (stop hitting) if the sum of the cards is higher than 17 (J, Q and K are each worth 10 points, A is worth 1 or 11 points). The aim is to get as close to 21 as possible. If you go over you are 'bust' (you have lost).

A basic blackjack program is shown below (and is provided electronically).

Study the code and try to understand what is happening in the program, before attempting to modify it.

```
1  import random
2
3  def getValue(card):
4      try:
5          return int(card)
6      except:
7          if card == "J" or "Q" or "K":
8              return 10
9          else:
10             return 11
11
12 print("Automatic Blackjack Player\n")
13
14 games = 0
15 gameOver=False
16
17 while not gameOver:
18
19     deck = ["A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]
20     random.shuffle(deck)
21
22     hand = []
23     score = 0
24
25     while score < 17 and len(deck) != 0:
26         card = deck.pop()
27         hand.append(card)
28         score = score + getValue(card)
29
30     if score == 21:
31         print("Blackjack!")
32         games += 1
33     if score < 21:
34         print("You have scored " + str(score))
35         games += 1
36     else:
37         print("Uh oh you have gone bust!")
38         games += 1
39
40     print("Your cards were " + hand + "\n")
41
42     if len(deck) < 1:
43         gameOver=True
44
45 number_of_games = games
46 print("\nYou played " + str(number_of_games) + " games")
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## SECTION A

**A | 1**


Initially, the program can correctly add the values of a hand, but crash when it tries to print the hand.

Explain why, and fix the issue electronically.

2021

Program updated ☐

A 2

Log Education  tly the program gets stuck in an infinite loop.

Explain why, and stop your electronic program from getting stuck in t

.....

.....

.....

Program updated ☐

A | 3

Explain the purpose of the code `* 4` when creating the deck variable  
why is it good practice to create the deck in this way?

A 4

There is a logic error in the `getValue()` function – all picture cards are identified below where the error occurs in this function, and find

.....

.....

Program updated ☐

A | 5

If you hit Blackjack, a message appears stating that you have gone bust. State the line number where the error occurs, and fix your program accordingly.

Program updated ☐

In addition to the number of games played, it would be useful to

49. How many times you scored higher than 17 (and how many times you went bust).

Change the variable `games` to a list. It should store [number of bluffs, number of busts] in that order.

The program should print out the percentage of the times that each

Program updated ☐

## SECTION B

B | 1

In addition to the number of games played, it would be useful to know how many times you scored higher than 17 (and how many times you went bust).

Change the variable `games` to a list. It should store [number of bluffs, number of busts] in that order.

The program should print out the percentage of the times that each

Program updated ☐

**COPYRIGHT  
PROTECTED**



**B 2**

In blackjack, the ace can either have the value 11, or the value 1. As the program only sees it as the value 11.

Change your program so that instead of going over 21 by counting it counts it as 1 instead, before continuing as normal.

Program updated ☐

**B 3**

In order to see how the game is in terms of the edge that the house has to run the game for a much longer period of time and get the computer to simulate a million hands. This is called modelling.



Change the number of decks to 500 – don't print out each hand, just the number of hands won with blackjacks, hands lost with blackjacks, hands won with a total < 17 and hands lost with a total >= 17 and total busts.

Program updated ☐

INSPECTION COPY



**COPYRIGHT  
PROTECTED**



## EXERCISE 10 – CONNECT FOUR

Connect Four is a two player logic game in which players take it in turns to drop a coloured piece of plastic into a grid, until one player has four colours matched in a row, column or diagonal.

To represent this grid on a computer, we can use **two-dimensional arrays**. Then, each value in the array can either be an 'R' or a 'B' depending on whether the tile in that space is red or black.

**Array:** A data type that can hold multiple values of the same data type.

A program that performs the logic needed to play the game is shown below (and is in the file `connectfour.py`). Study the code and try to understand what is happening in the program, before attempting the tasks.

```

1  def draw(grid):
2      print("")
3      print("1 2 3 4 5") # Print column headers
4      print("| | | | |")
5      print(grid[0][0], grid[0][1], grid[0][2], grid[0][3],
6            grid[1][0], grid[1][1], grid[1][2], grid[1][3],
7            grid[2][0], grid[2][1], grid[2][2], grid[2][3],
8            grid[3][0], grid[3][1], grid[3][2], grid[3][3],
9
10 def add_piece(grid, column, row, player):
11     if player == 1:
12         piece = "B"
13     else:
14         piece = "R"
15     grid[row][column] = piece
16     return grid
17
18 #- MAIN PROGRAM -----
19
20 board = [['0','0','0','0','0'],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],
21 won = False
22
23 draw(board)
24
25 while not won == True:
26
27     player = 1
28
29     print("It is player " + str(player) + "'s go.")
30     c_choice = int(input("Enter the column number. "))
31     r_choice = int(input("Enter the row number. "))
32
33     board = add_piece(board, c_choice, r_choice, player)
34
35     if player == 1:
36         player = 2
37     else:
38         player = 1
39
40     draw(board)
41
42     print(player + str(player) + " has won!")

```

**NOTE:** In this program, it will not close itself unless you have completed the tasks in the program press Ctrl+C.

**COPYRIGHT  
PROTECTED**



## SECTION A

A 1

The first error that you might notice is that the player does not change. Describe below why this happens, and then fix your program electronically.

.....

.....

Program updated ☐

A 2



The second row selection also does not work correctly. Describe below why this happens, and fix the problem on your electronic copy of the program.

.....

.....

Program updated ☐

A 3

Try entering a negative column, or a column greater than 5. Describe below what happens in the program when you try to enter a negative column number.

.....

.....

Add validation to the program after lines 30 and 31 to ensure that the input is valid. The program should repeatedly prompt the player to enter a valid column number.

Program updated ☐

A 4



The program currently overwrite other people's moves by choosing the space that is already occupied. Describe below the action you should take to stop this from happening. Update your electronic program to reflect this. *If the space is not empty, a message should be shown saying that the player has forfeited their turn.*

.....

.....

Program updated ☐

A 5

In the real game, you cannot put the game pieces anywhere – they can only be placed on the bottom row, or on top of another existing piece.

Instead of asking for a row number, it makes more sense to only ask the user to place the piece in the lowest empty cell on the board. Change your program to require the user to enter a column number and places a piece in the lowest free space in that column.

By modifying the `place_piece()` function, add verification to only allow moves if there is at least one empty space. If a move is invalid, you should show a message and the next player take their turn.



Program updated ☐

**COPYRIGHT  
PROTECTED**



## SECTION B

B 1

The grid in Connect Four actually uses seven columns and six rows.

Change the draw() function to allow *any* sized grid to be drawn. *The should be calculated from the grid parameter.*

Test that it works by changing line 20 to:

```
board = [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]]
```

*You may assume that the number of rows or columns will not be greater than 10.*



B 2

Change the program so that it asks the user to enter the size of the grid and then initialises the board correctly.

Program updated ☐

B 3

Your program currently runs forever. Describe the reason for this.

.....  
.....

To fix this, some checking needs to be put in place to determine who has won the game. Write a function check\_winner() that checks whether there are three in a row *horizontally or vertically*.

Use this function directly after adding a piece to stop the program if there is a winner (horizontally or vertically).



For two additional marks, extend this to check for upward diagonal winning lines, and then extend this to check for downward diagonal winning lines.

Program updated ☐



INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## Answers

In theory there are an almost infinite number of ways to program a solution to a problem. The following answers you should understand they are one way of the many ways to solve a problem.

### EXERCISE 1 - NUMBER GAME

A 1

Two marks (one for describing the issue, one for modifying the code as described).  
Lines 4 and 5 need to be added for the function guess() – they must be indented so that they belong to the function.



```
def guess():  
    num = input("Please enter your guess")  
    return num
```

A 2

One mark for modifying the code as described:

The problem is that the prompt doesn't provide a colon or a space for the user to type their answer. This looks confusing, but can be fixed simply by changing the prompt.

```
num = input("Please enter your guess: ")
```

A 3

Two marks for modifying the code as described:

Award marks for:

- adding a separator below the first one
- adding a blank line at the end



```
print("Welcome to the number guessing game")  
print("=====")  
print("The objective is to guess the number I'm thinking of")  
print("I will give you clues after your first guess")  
input()
```

A 4

One mark for modifying the code as described:

The change is on line 4 and involves using the int() function which converts the input to an integer.

```
num = int(input("Please enter your guess: "))
```

A 5

Three marks for modifying the code as described.

Award marks for:

- setting up the new variable outside the loop
- setting the loop condition correctly
- indenting the correct contents of the loop



```
numGuessed = 0  
while numGuessed != secretNumber:  
    numGuessed += 1  
    if numGuessed < secretNumber:  
        print("Guess is too low, guess higher!")  
    else:  
        print("Guess is too high, guess lower!")
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



**A 6**

Three marks (one for describing the issue, two for modifying the code)

Description: When the user guesses correctly it prints out that their program simply exits (which makes it almost look like a bug/error).

Award marks for:

- changing the 'else' to an 'elif' and providing the correct condition
- adding the new else clause and printing out a suitable message

```
numGuessed = 0
while numGuessed != secretNumber:
    numGuessed = guess()
    if numGuessed < secretNumber:
        print("Guess is too low, guess higher!")
    elif numGuessed > secretNumber:
        print("Guess is too high, guess lower!")
    else:
        print("Congratulations, you guessed correctly")
```

**B 1**

Three marks for modifying the code as described.

Award marks for:

- initialising the number of guesses outside the loop
- incrementing it in the correct place
- printing it out at the end in a suitably formatted message

```
numGuessed = 0
guesses = 0
while numGuessed != secretNumber:
    numGuessed = guess()
    guesses += 1
    if numGuessed < secretNumber:
        print("Guess is too low, guess higher!")
    elif numGuessed > secretNumber:
        print("Guess is too high, guess lower!")
    else:
        print("Congratulations, you guessed correctly")
```

**B 2**

Two marks for modifying the code as described. Note that the solution is possible.

Award marks as follows:

- 1 mark if they have solved it without a while loop (i.e. it just works)
- 2 marks if they have solved it using a while loop with the correct condition
- No marks should be given if the condition is incorrect

```
while num <1 or num >100:
    num = int(input("Your number must be in the range 1 to 100: "))
```

**COPYRIGHT  
PROTECTED**



<b>B</b>	<b>3</b>
----------	----------

Three marks for modifying the code as described. Note that the solution is not necessarily the only possible.

Award marks for:

- the exception being handled and the user asked to re-enter
- a working loop condition, meaning that it will carry on until
- the entire logic being correct, meaning that it will always return a number between 1–100 inclusive from the guess() function.

```
def guess():
    num = input("Please enter your guess: ")
    numEntered = False
    while not numEntered:
        try:
            num = int(num)
            while num <1 or num >100:
                num = int(input("Your number must be in the range 1-100: "))
            numEntered=True
        except:
            num = input("You must enter a number, try again: ")
    return num
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## EXERCISE 2 - ROCK, PAPER, SCISSORS

INSPECTION COPY

A 1

Two marks (one for describing the issue, one for modifying the code as follows)  
There is a missing colon (:) at the end of the function definition. Line 1:

```
def printRules():
```

A 2

Two marks (one for describing the issue, one for modifying the code as follows)  
There is an error with the indentation on line 5. The function should



```
def printRules():  
    print("The computer will think of either rock, paper or scissors")  
    print("You will enter r for rock, p for paper, s for scissors")  
    print("The computer will reveal it's choice and you will choose")  
    print()
```

A 3

Two marks (one for describing the error, one for modifying the code as follows)  
The error is on line 11. The correct code should be:

```
if computerChoice == 0:
```

A 4

Two marks (one for describing the issue, one for modifying the code as follows)  
The closing bracket is missing. The code should be:

```
print("The computer chose: Paper")
```

A 5

Two marks (one for describing the issue, one for modifying the code as follows)  
The comparison must be with the string "r". By using just r, the program variable called r will instead compare the value of the variable r to the value of the variable r.



```
if choice == "r":
```

A 6

Two marks (one for describing the error, one for modifying the code as follows)  
The closing quotation mark is missing. The code should be:

```
print("It's a draw")
```

A 7

Two marks (one for describing the issue, one for modifying the code as follows)  
The opening and closing brackets are missing. These are needed because we need to call a function (printRules); the code should be:

```
printRules()
```

A 8

Two marks (one for describing the issue, one for modifying the code as follows)  
The library random needs to be imported at the start of the program



```
import random
```


**COPYRIGHT  
PROTECTED**



**A 9**

Three marks (one for describing the issue, two for modifying the code)

The code needs to be changed so that when the player chooses something the game decides who the winner is and prints out the result. The selection should now be:



```

if choice == "r":
    if computerChoice == 0:
        print("It's a draw")
    elif computerChoice == 1:
        print("Computer Wins!")
    else:
        print("Player Wins!")
elif choice == "p":
    if computerChoice == 0:
        print("Player Wins!")
    elif computerChoice == 1:
        print("It's a draw")
    else:
        print("Computer Wins!")
else:
    if computerChoice == 0:
        print("Computer Wins!")
    elif computerChoice == 1:
        print("Player Wins!")
    else:
        print("It's a draw")

```

**B 1**

Two marks for modifying the code as described:

Award marks for:

- changing the input and asking the user to re-enter
- putting it into an iterative statement. This change is made



```

while choice not in ["r","p","s"]:
    choice = input("You must enter r, p or s – try again")

```

**B 2**

Two marks for modifying the code as described:

Award one mark each for modifying lines 8 and 10 to convert the choice

```

choice = input("Enter r for rock, p for paper or s for scissors")
while choice not in ["r","p","s"]:
    choice=input("You must enter r, p or s – try again")

```

**B 3**

Two marks for modifying the code as described:

There needs to be an iterative statement to keep checking if they have finished the game. Award one mark each for replacing lines 1 and 2 shown below.



```

playAgain = True
while playAgain:
    playGame()
    if input("Would you like to play again(y/n): ") != "y":
        playAgain=False

```

**COPYRIGHT  
PROTECTED**



### EXERCISE 3 - TURTLE DRAWING

INSPECTION COPY

A 1

Two marks (one for describing the issue, one for modifying the code as described).  
There should be a new line added before line 21 to put the pen for the line.

```
turtle.pendown()
```

A 2



Two marks (one for describing the issue, one for modifying the code as described).  
The turtle needs to turn left again to be facing in the correct direction. There are two choices for this; the first is preferred. There are, of course, other choices for this; the first is preferred. There are, of course, other choices for this; the first is preferred. This change is made after line 20.

```
turtle.left(90)
```

or to set the heading of the turtle back to the same as it was at the start

```
turtle.setheading(0)
```

A 3

One mark for modifying the code as described:  
This involves changing line 23 (it was originally line 21).

```
drawSquare(90, GREEN)
```

A 4

Two marks for modifying the code as described:  
Award marks for:  
• adding a new line with the hex value for red

```
RED = "#FF0000"
```



- modifying the line to change the colour from blue to red

```
drawSquare(100, RED)
```

A 5

One mark for modifying the code as described:  
There are two main ways this can be achieved  
The first option (preferred) is to place the square more evenly inside the circle.  
This involves modifying lines 18–20 as follows:

```
turtle.forward(10)  
turtle.right(90)  
turtle.forward(10)
```

The second option is to draw the square slightly larger. This is by modifying line 21 (it was originally line 21).

```
drawSquare(95, GREEN)
```



**COPYRIGHT  
PROTECTED**



**B 1**

Two marks (one for naming the command, one for modifying the code)

This can be done using **`turtle.hideturtle()`**.

This should be added just before the **`turtle.exitonclick()`** command

```
turtle.hideturtle()
```

**B 2**

Three marks for modifying the code as described.

Award marks for:

- drawing the correct shape
- achieving this without taking the pen off the paper
- calculating the side lengths and angles correctly so that the



```
def drawHouse():
    turtle.left(90)
    turtle.forward(100)
    turtle.right(45)
    turtle.forward(70)
    turtle.right(90)
    turtle.forward(70)
    turtle.right(45)
    turtle.forward(100)
    turtle.right(135)
    turtle.forward(141)
    turtle.right(135)
    turtle.forward(100)
    turtle.right(135)
    turtle.forward(141)
    turtle.left(135)
    turtle.forward(100)
```



```
turtle.penup()
turtle.setheading(0)
turtle.pendown()
turtle.color(BLUE)
```

```
drawHouse()
```

**B 3**

Three marks for modifying the code as described:

Award marks for:

- drawing the correct shape
- using an appropriate loop the correct number of times
- using fill commands to add colour

The drawStar function might look as follows:



```
def drawStar():
    turtle.color('red', 'yellow')
    turtle.begin_fill()
    for i in range(36):
        turtle.forward(200)
        turtle.left(170)
    turtle.end_fill()
```

**COPYRIGHT  
PROTECTED**



## EXERCISE 4 - THE MONTY HALL PROBLEM

INSPECTION COPY

A 1

One mark for a valid description:

The '\n' symbol allows a single line of code to be written over multiple purposes.

A 2

Two marks (one for describing the issue, one for modifying the code as

The program crashes because you cannot concatenate a string with the following message:



```
choice = input("The is a goat behind door " + str(door) + " or " + str(otherDoor) + " switch to door " + str(otherDoor) + "? (y/n) ")
```

A 3

Two marks (one for describing the issue, one for modifying the code as

The program does not work as input is read as a string, not an integer. Converting the input to an integer.

```
choice = int(input("Door 1, 2 or 3? "))
```

Note: This means that the program will crash if something other than a number is entered. A try catch clause should be used, but it can be assumed that the input will be a number.

A 4

One mark for modifying the code as described:

The random module must be imported before `random.shuffle()`.

```
import random

door = ["goat", "goat", "car"]
random.shuffle(door)
```

A 5

One mark for a valid description:

Arrays (lists) in Python always start at 0; the -1 is to convert from the index of the data is stored.

B 1

Two marks (one mark identifying the construct, one for modifying the

This is easily done by wrapping everything in a FOR loop (construct

```
for games in range(10):
    door = ["goat", "goat", "car"]
    ...
```

B 2

One mark for modifying the code as described:

This can be done using the function `random.randint()`:

```
choice = random.randint(1,3)
```



COPYRIGHT  
PROTECTED



**B 3**

Two marks for modifying the code as shown:

Only award one mark if the function is not called twice with 1,000 games

```
import random

def montyHall(numGames,switch):

    won = 0
    lost = 0

    for game in range(numGames):

        if switch:
            choice = otherDoor

        if door[choice-1] == "car":
            won += 1
        else:
            lost += 1

    print("You won", won, " cars and", lost, " games")

montyHall(1000,True)
montyHall(1000,False)
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## EXERCISE 5 - CAESAR CIPHER

**A** **1**

One mark for the correct answer:

The shift value is **13**.

**A** **2**

Three marks (one mark for describing the error, one mark for identifying the error, one mark for modifying the code as described):

There is a syntax error in line 1 – a missing quotation mark.

```
letters = " abcdefghijklmnopqrstuvwxyz"
```

**A** **3**

One mark for correctly identifying line 2

One mark for correctly modifying the code

```
def letter_to_number(letter):
    letters = "abcdefghijklmnopqrstuvwxyz"
    # Find the number corresponding to the given letter
    # In this case a = 0, b = 1, c = 2, ..., z = 25
    number = letters.index(letter)
    return number
```

**A** **4**

One mark for correctly identifying input function

One mark for correct implementation of an 'input' function with an appropriate prompt

```
plaintext = input("Enter a message: ")
```

**A** **5**

One mark for correctly identifying the .lower() function

One mark for correctly using the .lower() function at the correct point in line 15

```
def shift(letter):
    n = letter_to_number(letter.lower())
    return number_to_letter( (n + 13) % 26 )
```

**B** **1**

One mark for correctly identifying jgnnq yqtnf as the encrypted string

Four marks for the new encrypt function

Award marks for:

- correct declaration of the function with two parameters
- creating and returning an appropriate variable for the encrypted string
- looping through each character in the string parameter
- appending the encrypted character to the encrypted string

```
def encrypt(string, offset):
    ciphertext = ""
    # Creates the cipher text one step at a time
    for letter in string:
        ciphertext += shift(letter.lower(), offset)
    return ciphertext
```

**COPYRIGHT  
PROTECTED**





**Trivia**

To do division, a computer will do repeated addition. In the case of  $9 \div 3$ , it adds 3 to 0 and compares with the result; it's less, so it adds 3 again. This time the comparison shows that we have  $9 = 3 + 3 + 3$  so the  $\text{DIV } 3$  is 3,  $9 \text{ MOD } 3$  is 0. Taking the same idea but using  $10 \div 3$ , you take 3 over so  $10 = 3 + 3 + 3 + 1$  making  $10 \text{ DIV } 3$  equal to 3, but  $10 \text{ MOD } 3$  is 1.

To solve the problem set using a negative number, you start with the divisor repeatedly to get the remainder. For example,  $-10 \div 3$  means starting with 0 and adding 3 repeatedly, thus  $0 + 3 = 3$ ,  $3 + 3 = 6$ ,  $6 + 3 = 9$ ,  $9 + 3 = 12$ ; therefore,  $-10 \text{ DIV } 3$  is 4 and  $-10 \text{ MOD } 3$  is -2.

**B 2**

mark for decrypted string this is a secret message

Three marks for the new decrypt function

Award marks for:

- correct declaration of the function with two parameters
- variable declared and returned by the function
- call to 'encrypt' that uses the appropriate parameter, negative

```
def decrypt(string, offset):
    plaintext = encrypt(string, -offset)
    return plaintext
```

**B 3**

Three marks for the modified code

Award marks for:

- opening the text file in read ('r') mode
- reading the file into a plaintext variable
- calling the encrypt method with the plaintext variable and an



```
file = open("plaintext.txt", 'r')
plaintext = file.read()
file.close()
ciphertext = encrypt(plaintext, 13)
```

**COPYRIGHT  
PROTECTED**



## EXERCISE 6 - CHECK DIGITS

In theory there are an almost infinite number of ways to program a solution to a problem. In the following answers you should understand they are one way of the many ways to program a solution.

**A 1**

One mark for correctly identifying line 7

One mark for the modified version of that line

```
actual_check_digit = int((sum-1)%10)
```

**A 2**

Two marks for identifying that the code that should multiply the even numbers by 3 instead of 1.



One mark for either changing the value in the IF statement from 0 to 1 or changing the contents of the IF part with the ELSE part

```
for x in range(len(unique_id)):
    if x%2 == 1:
        times_three.append(unique_id[x]*3)
    else:
        times_three.append(unique_id[x])
```

**A 3**

One mark for identifying the sum function

One mark for the correct code in place of lines 11-13

```
sum_new_digits = sum(times_three)
```

**A 4**

Three marks for the additional code after line 51

Award marks for:

- use of a loop
- correct calculation of either 13 as valid or not(13) as invalid
- error message appears when invalid data is entered



```
choice = input("Enter the ISBN number: ")
while len(choice) != 13:
    choice = input("Error - you must have 13 characters")
```

**A 5**

One mark for identifying the required input statement (other possibilities accepted)

One mark for additional code after the output

```
input("Press enter to quit. ")
```

**B 1**

Three marks for the modified code after line 34

Award marks for:

- removal of dashes from input string
- IF statement checking for four dashes, '978' start and 13 non-digits
- contents of IF and ELSE clauses correct as below

N.B. other solutions exist and should be given full credit if they would work

```
choice = input("Enter the ISBN number: ")
ISBN_split = choice.split("-")
ISBN_numbers = "".join(ISBN_split)
if choice.count("-") == 4 and ISBN_split[0] == "978" and len(ISBN_numbers) == 13:
    print("ISBNcheck() returns " + ISBNcheck(ISBN_numbers))
else:
    print("Invalid ISBN!")
```



**COPYRIGHT  
PROTECTED**




**B 2**

One mark for acknowledging that Booleans require less storage or are compare than strings One mark for acknowledging that using Booleans is more efficient than strings of working

Three marks for the modified code

Award marks for:

- modified return statements on lines 30 and 32
- use of an IF expression that acts on a Boolean value returned from line 29
- output displayed correctly in IF and ELSE clauses



```

if check_digit == actual_check_digit:
    return True
else:
    return False
...
if ISBNcheck(ISBN_numbers) == True:
    print("ISBNcheck() returns True")
else:
    print("ISBNcheck() returns False")

```


**B 3**

Four marks for the new code

Award marks for:

- correct definition of function with a single parameter
- implementation of IF, ELIF, ELSE structure
- correct values returned in all cases
- call to getCountry function inside a print statement


N.B. other solutions exist and should be given full credit if they would work



```

def getCountry(number):
    if number == "0" or number == "1":
        return "an English speaking country."
    elif number == "2":
        return "a French speaking country."
    elif number == "3":
        return "a Germany speaking country."
    elif number == "4":
        return "Japan."
    elif number == "5":
        return "the (former) USSR."
    ...
if choice.count("-") == 4 and ISBN_split[0] == "0" and len(ISBN_numbers) == 13:
    if ISBNcheck(ISBN_numbers) == True:
        print("ISBNcheck() returns True")
        print("The book is for " + getCountry(choice[-1]) + ".")
    else:
        print("ISBNcheck() returns False")
else:
    print("Invalid ISBN!")

```



INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## EXERCISE 7 - HANGMAN

In theory there are an almost infinite number of ways to program a solution to a problem. The following answers you should understand they are one way of the many ways to program a solution.

A 1

One mark for recognising that `=` is used instead of `==`  
One mark for modifying code on lines 18 and 19:

```
if user_guess == word[1]:
```

```
if guessed_word == word: ...
```

A 2



mark: `\n` is the newline character. It can be used to simulate pressing the enter key.  
mark: `\` allows a line of code to be split across multiple lines in the code editor.

A 3

One mark for identifying line 29  
One mark for the modified code for that line

```
elif lives < 1:
```

A 4

One mark: `'c'` is not recognised as the same character as `'C'`  
One mark for including `.lower` in line 12

```
user_guess = input("Guess a letter/word! (" + str(lives) + " lives remaining)\n").lower()
```

Alternatively, the user could force all strings to be uppercase but then followed through by changing the word list too.

A 5

One mark: connects two strings together into a larger string  
One mark: inserts the string `after every character` of the second string

B 1



mark: modifying code on line 25 to the following effect  
mark: if no additional lines are written, so line 25 is as below:

```
if guessed_word == word or list(user_guess) == word:
```

B 2

Seven marks for the modified code

Award marks for:

- importing random
- opening the file in read (r) mode
- splitting the file contents with `.split(",")`
- reading the contents of the file into wordlist
- choosing a random word
- converting the random word to lower case
- creating the correct mask for the words with `_` underscores

```
import random
```

```
...
```

```
file = open("wordlist.txt", 'r') # Open file
```

```
wordlist = file.read().split(",") # Split content
```

```
file.close() # Close file
```

```
randomWord = random.randint(1, len(wordlist) - 1)
```

```
word = list(wordlist[randomWord].lower()) # Don't  
every word to lowercase
```

```
guessed_word = list("_" * len(word))
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



**B 3**

One mark: declaring an array alongside other variable declarations

```
guessed_letters = []
```

One mark: appending guesses to the new array

One mark: only appending for letters not in the word

```
if not letter_in_word:  
    lives -= 1  
    guessed_letters.append(letter_guess)
```

One mark: modifying line 12 to include an output for the array's contents



```
letter_guess = input("Incorrect letters/words: " +  
str(guessed_letters) + "\n" \  
"Guess a letter/word! (" + str(lives) + \  
" lives remaining)\n").lower()
```



INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## EXERCISE 8 - PEG SOLITAIRE

In theory there are an almost infinite number of ways to program a solution to a problem. In the following answers you should understand they are one way of the many ways to program a solution.

A 1

One mark: arrays' elements in Python begin at 0 (not 1)  
One mark: modifying find function to remove values

```
def find(board, piece):
    for i in range(len(board)):
        for j in range(len(board[0])):
            if board[i][j] == piece:
                return i, j
```



A 2

One mark: this program makes use of many constant values/characters  
One mark: a variable could be used to store a space  
Two marks for modifying the code (below is a snippet, but all IF clause must be modified)  
Award marks for:

- declaring a variable that contains a space
- using that variable in lieu of "X"

```
SPACECHAR = " "

if direction == "r":
    if column + 2 < len(grid[0]):
        if grid[row][column + 2] == SPACECHAR:
            grid[row][column] = SPACECHAR
            grid[row][column + 1] = SPACECHAR
            grid[row][column + 2] = choice
        ...
```



A 3

Two marks for new code  
Award marks for:

- declaring the show procedure
- looping once per row
- outputting the row
- calling the procedure from line 10

```
def show(board):
    for row in board:
        print(row)
```

Equally acceptable alternative solution:

```
def show(board):
    for row in range(len(board)):
        for col in range(len(board[row])):
            print(board[row][col], end=" ")
        print()
```



INSPECTION COPY

COPYRIGHT  
PROTECTED



**A 4**

One mark: use of .lower on lines 10 and 11

```
choice = input("Enter a letter: ").lower()
direction = input("Enter a direction (u,d,l,r): ")
```

Alternatively, in every place at which a comparison is made between to see if it matches the uppercase letter, e.g.:

```
if direction == "r" or direction == "R":
```

**A 5**

One mark: nesting statements



Four marks for modified code

Award marks for:

- use of Boolean operator 'and'
- code modified with no errors

```
if direction == "r" and column + 2 < len(grid[0]) and grid[row][column + 1] != SPACECHAR:
    grid[row][column] = SPACECHAR
    grid[row][column + 1] = SPACECHAR
    grid[row][column + 2] = choice
```

```
if direction == "l" and column - 2 >= 0 and grid[row][column - 1] != SPACECHAR:
    grid[row][column] = SPACECHAR
    grid[row][column - 1] = SPACECHAR
    grid[row][column - 2] = choice
```

```
if direction == "u" and row - 2 >= 0 and grid[row - 1][column] != SPACECHAR:
    grid[row][column] = SPACECHAR
    grid[row - 1][column] = SPACECHAR
    grid[row - 2][column] = choice
```

```
if direction == "d" and row + 2 < len(grid) and grid[row + 1][column] != SPACECHAR:
    grid[row][column] = SPACECHAR
    grid[row + 1][column] = SPACECHAR
    grid[row + 2][column] = choice
```

**B 1**

One mark: there is no check that the adjacent square contains a space

Four marks for modified code

Award marks for:

- checking 'column+1' location for a move to the right
- checking 'column-1' location for a move to the left
- checking 'row+1' location for a move down
- checking 'row-1' location for a move up

Code below includes modifications for a move right:



```
if direction == "r" and column + 2 < len(grid[0]) and grid[row][column + 1] != SPACECHAR:
    grid[row][column] = SPACECHAR
    grid[row][column + 1] = SPACECHAR
    grid[row][column + 2] = choice
```

Alternatively, an additional IF statement can be used (if A5 has not b

**COPYRIGHT  
PROTECTED**



**B 2***Eleven marks for modified code*

Award marks for:

- IF statement to check for 'save'
- opening the file in write (w) mode
- writing the number of rows and columns to the file first
- nested loop structure
- game state correctly written to file with a carriage return character on a new line
- ELIF statement to handle 'load'
- opening the file in read (r) mode
- grid array initialisation
- nested loop structure
- grid properly populated
- code functions as normal for any game play moves



```

...
choice = input("Enter a letter: ").lower()
if choice == "save":
    file = open("game.txt", 'w')
    file.write(str(rows)+"\n"+str(columns)+"\n")
    for row in grid:
        for item in row:
            file.write(item+"\n")
    file.close()
elif choice == "load":
    file = open("game.txt", 'r')
    rows = int(file.readline().strip())
    columns = int(file.readline().strip())
    grid = []
    for row in range(rows):
        grid.append([])
        for column in range(columns):
            grid[row].append(file.readline().strip("\n"))
    file.close()
...
direction = input("Enter a direction (u,d,l,r) ")
row, column = find(grid, choice)
...

```

**B 3***Three marks for modified code*

Award marks for:

- declaration and initialisation of variables as below
- use of 'chr' to get the character from the ASCII code
- incrementation of 'count'

```

# Creating the board array
rows = 6
columns = 4
grid = []
count = 97

for i in range(rows):
    grid.append([])
    for j in range(columns):
        if count != 98:
            grid[i].append(chr(count))
        else:
            grid[i].append(spaceChar)
        count += 1

```

**COPYRIGHT  
PROTECTED**



Three marks for additional code

Award marks for:

- determining the number of empty spaces on the board
- comparing this with the number of available places minus 1
- terminating the main loop if these values match

N.B. other solutions exist and should be given full credit if they would

```
# Function to check if the game has been won.
def has_won(board):
    board_string = ""
    # Make a string of the whole board
    for row in board:
        for item in row:
            board_string += item

    # Count the total number of spaces, if it is equal to
    # width times the height, then there is one of each letter

    if board_string.count(spaceChar) == len(board):
        return True
    else:
        return False

# Main game loop
while not has_won(grid):
    show(grid)
    choice = input("Enter a 1-9: ").lower()
    ...
```

Another solution to this problem might be:

```
# Function to check if the game has been won.
def has_won(board):
    peps = 0
    # Parse the board
    for row in board:
        for item in row:
            if item != SPACECHAR:
                peps += 1
    if peps == 1:
        return True
    else:
        return False

# Main game loop
...
```

INSPECTION COPY

COPYRIGHT  
PROTECTED



## EXERCISE 9 - BLACKJACK HANDS

In theory there are an almost infinite number of ways to program a solution to a problem. The following answers you should understand they are one way of the many ways to program a solution.

A 1

One mark: you cannot concatenate a string with an array OR the array with a string.  
One mark: correct use of `str()`

```
print("Your cards were " + str(hand) + "\n")
```

A 2

One mark: the length of the deck array never reaches zero...

One mark: ...because it is re-populated at the beginning of each loop

One mark: moving the code to populate and shuffle the deck to **before** the loop

```
...
deck = ["A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]
random.shuffle(deck)

while not gameOver:

    hand = []
    score = 0

    ...
```

A 3

One mark: multiplying a list by an integer creates copies of each element

One mark: manually creating copies of each element by repeating code

A 4

One mark: the IF statement on line 33 is incorrect as it does not reset the score

One mark: modifying a copy

```
if card == "J" or card == "Q" or card == "K":
```

A 5

One mark: error exists in line 33

One mark: IF on line 33 replaced with ELIF

```
if score == 21:
    print("Blackjack!")
    games += 1
elif score < 21:
    print("You have scored " + str(score))
    games += 1
else:
    print("Uh oh, you have gone bust!")
    games += 1
```

INSPECTION COPY

COPYRIGHT  
PROTECTED



**B 1**

Three marks for additional code

Award marks for:

- games is now a list rather than a variable
- elements within games are incremented within correct IF, ELIF, ELSE clause
- calculations correct to return percentages

```

games = [0,0,0] # First element is blackjacks, second is scored > 17, and third is bust.

...

if score == 21:
    print("Blackjack!")
    games[0] += 1
elif score < 21 and score > 17:
    print("You have scored " + str(score))
    games[1] += 1
else:
    print("Uh oh, you have gone bust!")
    games[2] += 1

...

number_of_games = sum(games)
print("\nOut of " + str(number_of_games) + " game"
print("- hit Blackjack " + str(games[0]) + " time"
print("  scored (> 17) " + str(games[1]) + " time"
print("  went bust " + str(games[2]) + " times ("
print(str(100*games[0]/number_of_games) + "% of the time"
print(str(100*games[1]/number_of_games) + "% of the time"
print(str(100*games[2]/number_of_games) + "% of the time")

```

**B 2**

Three marks for additional code

Award marks for:

- IF statement to determine both presence of ace **and** score a
- subtraction of 10, only in these circumstances
- code capable of handling multiple aces, including those dealt (implemented here in nested IF clause)

N.B. other solutions exist and should be given full credit if they would

```

acesCounted = 0
while score < 17 and len(deck) != 0:
    card = deck.pop()
    hand.append(card)
    score = score + getValue(card)
    if score > 21 and "A" in hand:
        if acesCounted < len(hand) - 1:
            score -= 10
            acesCounted += 1

```

INSPECTION COPY

**COPYRIGHT  
PROTECTED****Zig  
Zag  
Education**

Award marks for:

- modification of deck declaration to multiply by 500
- player and computer score both to reset to zero on each iteration
- computer to draw cards in the same way that player draws cards
- updated and aces being tracked
- selection structure to handle player score being higher
- selection to handle computer score being higher
- selection to handle player and computer scores being the same
- selection to handle player being bust
- calculation summary to include games won vs lost
- output to include all percentage calculations (won with blackjack with  $\geq 17$ , lost with  $\geq 17$ , bust)
- all output correct (format does not need to be identical)

```

deck = ["A", "2", "3", "4", "5", "6", "7", "8", "9",
...

while not gameOver:

    playerHand = []
    playerScore = 0
    acesCounted = 0

    while playerScore < 17 and len(deck) != 0:
        card = deck.pop()
        playerHand.append(card)
        playerScore += getValue(card)
        if playerScore > 21 and "A" in playerHand:
            if acesCounted < playerHand.count("A"):
                playerScore -= 10
                acesCounted += 1

    computerHand = []
    computerScore = 0
    acesCounted = 0

    while computerScore < 17 and len(deck) != 0:
        card = deck.pop()
        computerHand.append(card)
        computerScore += getValue(card)
        if computerScore > 21 and "A" in computerHand:
            if acesCounted < computerHand.count("A"):
                computerScore -= 10
                acesCounted += 1

    if playerScore == 21:
        if playerScore == computerScore:
            gamesLost[0] += 1
        else:
            gamesWon[0] += 1
    elif playerScore > 21:
        if playerScore < 21 and computerScore >= playerScore:
            gamesLost[1] += 1
        else:
            gamesWon[1] += 1
    else:
        gamesLost[2] += 1

```

(continues on the next page)

INSPECTION COPY

COPYRIGHT  
PROTECTED

(from previous page)

```
if len(deck) < 1:
    gameOver=True

number_of_games = sum(gamesWon) + sum(gamesLost)
print("\nOut of " + str(number_of_games) + " game
print("- hit Blackjack " + str(gamesWon[0]) + " t
str(100*gamesWon[0]/number_of_games)[0:5] + "% of
" +str(gamesWon[1]*100/(gamesWon[0]+gamesLost[0])
print("went bust (>16) " + str(gamesWon[1]) + " ti
str(gamesWon[1]/number_of_games)[0:5] + "% of
+str(gamesWon[1]*100/(gamesWon[1]+gamesLost[1])
print("- went bust " + str(gamesLost[2]) + " time
str(100*gamesLost[2]/number_of_games)[0:5] + "% o

input("Press enter to exit.")
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## EXERCISE 10 - CONNECT FOUR

In theory there are an almost infinite number of ways to program a solution to a problem. The following answers you should understand they are one way of the many ways to program a solution.

A 1

One mark: player is set to 1 at the beginning of each loop (so will never be 0)  
One mark: assignment into player variable is done before loop

```
board = [[0,0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]
won = False
player = 1
while not won == True:
    draw(board)
    print("It is player " + str(player) + "'s go.")
    ...
```

A 2

One mark: arrays begin with an index of 0, not 1 OR column headers and row headers indices will be 0-4

One mark: subtracting 1 from each of row and column (this could be done in one line) and the mark can be awarded if 1 is subtracted anywhere between empty and full

```
def add_piece(grid, column, row, player):
    if player == 0:
        piece = "B"
    else:
        piece = "R"
    grid[row-1][column-1] = piece # Changed here
    return grid
```

A 3

One mark: the choice loops around for negative values. For example, if column is -1, it gives the second-to-last column.

Two marks for new code

Award marks for

- loops that repeat until data in valid range is entered
- error messages that only appear when invalid data is entered

```
while not c_choice > 0 and c_choice < 6:
    print("There has been an error")
    c_choice = int(input("Enter the column number. "))

while not r_choice > 0 and r_choice < 5:
    print("There has been an error")
    r_choice = int(input("Enter the row number. "))
```

A 4

One mark: description of if statement that the selected space is empty

Two marks for new code

- IF statement to check that the space is either empty or not empty
- IF and ELSE clauses correctly either place the piece or display error message

```
if grid[row-1][column-1] == "0":
    grid[row-1][column-1] = piece
else:
    print("Space is already taken! Turn is being passed")
```

**COPYRIGHT  
PROTECTED**




**A 5**

Three marks for new code

Award marks for

- reference to row no longer included as a parameter
- either loop from the bottom of the column upward until a space is found, or from the top downward until a counter is found, then move
- checking whether the column is full



```
def add_piece(grid, column, player):

    if player == "X":
        piece = "X"
    else:
        piece = "R"

    # Find the lowest row that can fit the piece.
    # fill up from the bottom, we can count upward
    row = -1
    for i in range(len(board)):
        if grid[i][column-1] == "0":
            row = i

    # If row = -1 then a space could not be found,
    # should be skipped. Otherwise we can insert


    if row == -1:
        print("Column is full! Skipping turn")
    else:
        grid[row][column-1] = piece
```

**B 1**

Three marks for new code

Award marks for

- loop to add correct number of column headers based on array
- nested loop to build the board itself
- row headers added for all rows using variable that increments



```
def draw(grid):
    print("")

    # Build the column reference row
    headers = " 1"
    for i in range(2, len(grid[0]) + 1):
        headers += " " + str(i)
    print(headers)

    # Build the actual board
    for x in range(1, len(grid)):
        row = str(grid[x][0])
        for y in range(1, len(grid[0])):
            row += " " + str(grid[x][y])
        row += " row " + str(x+1)
        print(row)
```

**COPYRIGHT  
PROTECTED**

One mark: updating validation routines to account for the size of the board

```
while not (c_choice > 0 and c_choice <= len(board)):
    print("There has been an error")
    c_choice = int(input("Enter the column number. "))

while not (r_choice > 0 and r_choice <= len(board)):
    print("There has been an error")
    r_choice = int(input("Enter the row number. "))
```

Four marks for:



- loop in place to handle invalid dimensions (outside range 0-9)
- prompting the user for both rows and columns
- error message for invalid row or column entry
- nested loop to construct the board if valid data is entered

```
rows=0
cols=0
board=[]
while rows<1 or cols<1 or rows>9 or cols>9:
    try:
        rows=int(input("How many rows would you like? "))
        cols=int(input("How many columns would you like? "))
        if rows<1 or cols<1 or rows>9 or cols>9:
            print("Rows and columns must be in the range 1-9")
        else:
            for i in range(rows):
                board.append([])
                for j in range(cols):
                    board[i].append("0")
    except:
        rows=0
        cols=0
        board=[]
        print("Invalid entry, please try again. ")
```



INSPECTION COPY

**COPYRIGHT  
PROTECTED**





One mark: the won variable is never set to true

Eight marks for new code

Award marks for

- declaration of new function with appropriate return value
- separate checking for a victory along a horizontal, vertical or orientations, even if incorrect
- horizontal checking involves either a consecutive counter with a counter that is checked for a value of 4
- vertical checked in a similar way
- two marks for upward diagonal code (one can be awarded for successful) for a nested loop that iterates through both rows and columns
- two marks for downward diagonal code (one can be awarded for successful) for a nested loop that iterates through both rows and columns



```
def check_winner(grid):
    # Checking the rows
    for row in grid:
        row_string = " ".join(row)
        if "BBBB" in row_string or "RRRR" in row_string:
            return True

    # Checking the columns
    for i in range(len(grid[0])):
        column = [row[i] for row in grid]
        column_string = " ".join(column)
        if "BBBB" in column_string or "RRRR" in column_string:
            return True

    # Checking the upward diagonals
    diagonals=[]
    for i in range(len(grid)):
        diagonal.append("")
        for j in range(i, -1, -1):
            if i-j < len(grid[0]):
                diagonals[i]+=grid[j][i-j]
    for i in range(1, len(grid[0])):
        diagonals.append("")
        for j in range(len(grid)-1, -1, -1):
            if i-j+len(grid)-1 < len(grid[0]):
                diagonals[i+len(grid)-1]+=grid[j][i-j+1]
    for i in range(len(diagonals)):
        if "BBBB" in diagonals[i] or "RRRR" in diagonals[i]:
            return True

    # Checking the downward diagonals
    diagonals=[]
    for i in range(len(grid)-1, -1, -1):
        diagonals.append("")
        for j in range(len(grid)-i):
            diagonals[len(grid)-1-i]+=grid[i+j][j]
    for i in range(1, len(grid[0])):
        diagonals.append("")
        for j in range(len(grid)):
            if i+j < len(grid[0]):
                diagonals[i+len(grid)-1]+=grid[j][i+j]
    for i in range(len(diagonals)):
        if "BBBB" in diagonals[i] or "RRRR" in diagonals[i]:
            return True

    return False
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



# PYTHON QUICK HELP SHEET

Variable assignment:

Variable	Constant
<code>x = 16</code>	<code>MYCONSTANT = 16</code> #no constants in Python, but convention is to use

Arithmetic operations:

Add	Subtract	Multiply	Divide	Power
<code>1+2 = 3</code>	<code>5-4 = 1</code>	<code>3*4 = 12</code>	<code>10/4 = 2.5</code>	<code>3**2 = 9</code>

Boolean operators (in conditions):

AND	OR	NOT
<code>True and False</code>	<code>False or True</code>	<code>not True</code>

Boolean (comparison) operators:

Less than	Less than or equal to	Greater than	Greater than or equal to
<code>&lt;</code>	<code>&lt;=</code>	<code>&gt;</code>	<code>&gt;=</code>
<code>3&lt;4</code>	<code>3&lt;=3</code>	<code>4&gt;3</code>	<code>4&gt;=4</code>

The three standard control statements:

IF	FOR	WHILE
<pre>x = 1 if x == 1:     print("x = 1") elif x == 2:     print("x = 2") else:     print("x != 1 or 2")</pre>	<pre>for i in range(10):     print(i)</pre>	<pre>x = 3 while x &gt; 0:     print(x)     x = x - 1</pre>

For each statement – (iterates through the list allowing you to directly look at the list)

FOR EACH
<pre>y = [1,2,3,4,5] for item in y:     print(item)</pre>

String manipulation:

Function	Description
<code>[x]</code>	Returns the character in the x'th slot <pre>st = "Hello World" print(st[0]) &gt;&gt; H</pre>
<code>[x:y]</code>	Returns the string starting at the x'th slot, ending before the y'th slot <pre>st = "Hello World" print(st[0:2]) &gt;&gt; He</pre>
<code>len</code>	Returns the length of the string <pre>st = "Hello World" print(len(st)) &gt;&gt; 12</pre>
<code>+</code>	Concatenates two strings <pre>print("Hello" + "World") &gt;&gt; HelloWorld!</pre>

INSPECTION COPY

COPYRIGHT  
PROTECTED



Character manipulation:

Character to ASCII code	ASCII code to character
ord("a") (= 97)	chr(97) ("a")

Type conversion:

to Boolean	to String	to Integer	to Real Number
bool(1) (= True)	str(5) ("5")	int("2") (= 2)	float("5") (= 0.5)

Printing to screen

Print
print("Hello", "world!") >> Hello world!

File handling:

Reading	Writing
file = open("a.txt",'r') x = file.read() file.close()	file = open("a.txt",'w') file.write("Hello!") file.close()

List comprehension:

Examples
x = [i for i in range(10)] y = [i**2 for i in range(10)] z = [i**2 if i % 2 == 0 else i for i in range(10)]

List manipulation:

Function	Description
[x]	Returns the element in the x'th slot li = [1,2,3,4,5] print(li[0]) >> 1
[x:y]	Returns an array of the elements in the range from x to y li = [1,2,3,4,5] print(li[0:2]) >> [1,2]
[-x:]	Returns the last x elements in the array li = [1,2,3,4,5] print(li[-1:]) >> [5]
len	Returns the length of the list li = [1,2,3,4,5] print(len(li)) >> 5
+	Joins two lists [1,2] + [3,4] = [1,2,3,4]

INSPECTION COPY

COPYRIGHT  
PROTECTED



Other commands:

Function	Syntax	
def	def functionName(p1, p2....):	<pre>def addUp(num1, num2):     return num1+num2 print(addup(2,4)) &gt;&gt;&gt; 6</pre>
range	range (start, end, step):	<pre>for i in range(0,6):     print(i) &gt;&gt;&gt; 0 2 4</pre>
import	import libraryName	<pre>import random</pre>
try...except	<pre>try:     Code to try except:     Code when exception generated</pre>	<pre>try:     return int(card) except:     if card == "J" or "Q":         return 10     else:         return 11</pre>

External libraries:

Library	Description
random	Provides functions to generate random numbers
turtle	Provides a separate screen which you can use to control a virtual turtle

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



# PYTHON COMMON ERROR GUIDE

Programming can be frustrating – especially when there are errors in your code that you are not used to – if there is an obvious error in your program it will not run, and often states that it does not, however, deal with what are known as logic errors. These are errors in your program that are not in the way that you actually intend.

The guide quickly runs through the way to interpret common error messages, gives examples of errors, and shows how you can fix them.

## 1 – SYNTAX (PRINT STATEMENTS)

### CODE

Consider the following erroneous code:

```
print(Hello world!)
```

### ERROR MESSAGE

```
File "test.py", line 1
print(Hello world!)
^
```

SyntaxError: invalid syntax

### DESCRIPTION OF THE ERROR

We are trying to print a string – but the interpreter does not recognise what we have written by wrapping what we have written in quotation marks.

### FIXED CODE

```
print("Hello world!")
```

## 2 – SYNTAX (IF STATEMENTS)

### CODE

Consider the following erroneous code:

```
x = 1
if x = 1:
    print("Hello world")
```

### ERROR MESSAGE

The error message shown below is printed to the screen.

```
File "test.py", line 2
if x = 1:
^
```

SyntaxError: invalid syntax

### DESCRIPTION OF THE ERROR

A syntax error occurs when Python does not recognise what has been written. In this case, the operator `=` is used instead of `==` to compare the variable `x` to the value `1`. This can be fixed by using the correct sign for equality.

### FIXED CODE

```
x = 1
if x == 1:
    print("Hello world")
```

INSPECTION COPY

COPYRIGHT  
PROTECTED



### 3 – LOGIC (IF STATEMENTS)

#### CODE

Consider the following erroneous code. It is meant to add 1 to the variable x, and then print the value of x.

```
x = 1
```

```
if x == 1:  
    x = 2  
if x == 2:  
    x = 3  
if x == 3:  
    x = 4  
print(x)
```

#### ERROR MESSAGE

A logic error occurs when the program runs without crashing, but the intended function does not work correctly. Because this is a logic error, there is no error message. The program actually prints the value 2.

#### DESCRIPTION OF THE ERROR

This can be fixed using ELIF statements. In this case it means that the first IF statement is executed, and then the program will jump to the end of the IF statement.

#### FIXED CODE

```
x = 1  
  
if x == 1:  
    x = 2  
elif x == 2:  
    x = 3  
elif x == 3:  
    x = 4  
  
print(x)
```

### 4 – SYNTAX (IF STATEMENTS)

#### CODE

Consider the following erroneous code:

```
x = 1  
if x == 1  
    print("Hello world")
```

#### ERROR MESSAGE

The error message shown below is printed to the screen.

```
FILE "TEST.PY", LINE 2  
  IF x = 1  
    ^
```

SYNTAXERROR: INVALID SYNTAX

#### DESCRIPTION OF THE ERROR

The error occurs because a colon is missing.

#### FIXED CODE

```
x = 1  
if x == 1:  
    print("Hello world")
```

INSPECTION COPY

COPYRIGHT  
PROTECTED



## 5 – SYNTAX (FOR LOOPS)

### CODE

Consider the following erroneous code:

```
for x in range(12)
    print(x)
```

### ERROR MESSAGE

The error message shown below is printed to the screen.

```
FILE "TEST.PY", LINE 1
FOR X IN RANGE(12)
^
SYNTAXERROR: INDENT SYNTAX
```

### DESCRIPTION OF THE ERROR

The error occurs because a colon is missing.

### FIXED CODE

```
for x in range(12):
    print(x)
```

N.B. This error is exactly the same as Error 4 – colons should be used after IF, ELIF, ELSE.

## 6 – LOGIC (FOR LOOPS)

### CODE

Consider the following erroneous code. You wish to print the numbers 1–10.

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

### ERROR MESSAGE

Because this is a logic error, there is no error message.

The program prints: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

### DESCRIPTION OF THE ERROR

The error occurs because FOR loops start at 0 if the start is not specified, and Python uses the upper bound.

### FIXED CODE

```
for x in range(1,11):
    print(x)
```

INSPECTION COPY

COPYRIGHT  
PROTECTED



## 7 – LOGIC (DEF STATEMENTS)

### CODE

Consider the following erroneous code. You wish to increment x by one and print the result.

```
def f(x):  
    result = x + 1  
  
x = 1  
print( f(x) )
```

### ERROR MESSAGE

Because this is a logic error, there is no error message.  
The program returns `None`

### DESCRIPTION OF THE ERROR

The error occurs because no value is returned from the function.

### FIXED CODE

```
def f(x):  
    return x + 1  
  
x = 1  
print( f(x) )
```

## 8 – TYPE (INPUT STATEMENTS)

### CODE

Consider the following erroneous code:

```
x = input("Enter a number")  
  
if x < 10:  
    print("Less than 10")  
else:  
    print("Greater than 9")
```

### ERROR MESSAGE

The error message is shown below.

```
TRACEBACK (MOST RECENT CALL LAST):  
  FILE "TEST.PY", LINE 3, IN <MODULE>  
    IF x < 10:  
TypeError: UNORDERABLE TYPES: STR() < INT()
```

### DESCRIPTION OF THE ERROR

The user's input is stored as a string. When checking if the string is less than 10 and comparing a string to an integer, an error occurs.

### FIXED CODE

```
x = int(input("Enter a number: "))  
  
if x < 10:  
    print("Less than 10")  
else:  
    print("Greater than 9")
```

INSPECTION COPY

COPYRIGHT  
PROTECTED





## 9 – FILE (FILE HANDLING)

### CODE

Consider the following erroneous code:

```
file = open("test.txt")
file.write("Hello World! ")
file.close()
```

### ERROR MESSAGE

The error message is shown below:

```
TRACEBACK (MOST RECENT CALL LAST):
  FILE "TEST.PY", LINE 1, IN <MODULE>
    FILE = OPEN("TEST.TXT")
FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

### DESCRIPTION OF THE ERROR

This error has occurred because the file that the program is trying to open does not exist.

### FIXED CODE

Nothing needs to be changed in the program; a file named 'test.txt' needs to be in the current directory.

## 10 – I/O (FILE HANDLING)

### CODE

Consider the following erroneous code:

```
file = open("test.txt")
file.write("Hello World!")
file.close()
```

### ERROR MESSAGE

The error message is shown below.

```
TRACEBACK (MOST RECENT CALL LAST):
  FILE "TEST.PY", LINE 2, IN <MODULE>
    FILE.WRITE("HELLO WORLD!")
IO.UnsupportedOperation: not writable
```

### DESCRIPTION OF THE ERROR

This error has occurred because the file has been opened in a way that only allows it to be read.

### FIXED CODE

```
file = open("test.txt", 'w')
file.write("Hello World!")
file.close()
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## 11 – NAME (EXTERNAL LIBRARIES)

### CODE

Consider the following erroneous code:

```
print(math.sqrt(4))
```

### ERROR MESSAGE

The error message is shown below.

```
TRACEBACK (MOST RECENT CALL LAST):  
FILE "TEST.PY", LINE 1, IN <MODULE>  
PRINT(MATH.SQRT(4))  
NAMEERROR: NAME 'MATH' IS NOT DEFINED
```

### DESCRIPTION OF THE ERROR

This error has occurred because the external library `math` has not been imported.

### FIXED CODE

```
import math  
print(math.sqrt(4))
```

## 12 – INDEX (ARRAYS AND LISTS)

### CODE

Consider the following erroneous code:

```
x = []  
x[0] = 1  
x[1] = 2  
x[2] = 3
```

### ERROR MESSAGE

The error message is shown below.

```
TRACEBACK (MOST RECENT CALL LAST):  
FILE "TEST.PY", LINE 2, IN <MODULE>  
x[0] = 1  
INDEXERROR: LIST ASSIGNMENT INDEX OUT OF RANGE
```

### DESCRIPTION OF THE ERROR

This error has occurred because the size of the list has not been defined. Instead, to should use `.append()`

### FIXED CODE

```
x = []  
x.append(1)  
x.append(2)  
x.append(3)
```

INSPECTION COPY

**COPYRIGHT  
PROTECTED**



## 13 – LOGIC (ARRAYS AND LISTS)

### CODE

Consider the following erroneous code. It is meant to print out the elements in the list

```
x = [1,2,3,4]

for i in range(1,4):
    print(x[i])
```

### ERROR MESSAGE

As this is a logic error, no error message is printed. The program prints 2, 3, 4.

### DESCRIPTION OF THE ERROR

This error has occurred because the size of the list has not been defined. Instead, to add elements to a list, you should use `.append()`

### FIXED CODE

```
x = []
x.append(1)
x.append(2)
x.append(3)
```

## 14 – TYPE (STRINGS)

### CODE

Consider the following erroneous code. It is meant to print out the elements in the list

```
print("This is a number: " + 3)
```

### ERROR MESSAGE

```
TRACEBACK (MOST RECENT CALL LAST):
  FILE "TEST.PY", LINE 2, IN <MODULE>
    PRINT("THIS IS A NUMBER: " + 3)
  TypeError: CAN'T CONVERT 'INT' OBJECT TO STR IMPLICITLY
```

### DESCRIPTION OF THE ERROR

This error has occurred because you cannot concatenate a string with a number. You need to convert the number to a string.

### FIXED CODE

```
print("This is a number: " + str(3))
```

INSPECTION COPY

COPYRIGHT  
PROTECTED

