NAME:

A Level Computer Science

4.1 Programming

YEAR 11 LESSON

D.Ramduny-Ellis

Table of Contents

Learning objectives

- Explain the purpose and use of variables and data types.
- Trace and interpret the flow of control within programs using sequence, iteration and selection.
- Understand modular program design and function decomposition.
- Demonstrate problem-solving skills using a given skeleton code.
- Modify and extend existing code to add new features
- Test and debug programs to ensure they work correctly.

Starter

Figure 1 shows the same program in both pseudocode and Python.

- Line numbers are included but are not part of the program.

Figure 1

Pseudocode

```
1.   i ← 0
2.   y ← 1
3.   WHILE y > -1
4.       i ← i + 1
5.       y ← y - 2
6.       c ← i + y
7.       IF c > 3 THEN
8.           y ← y - 1
9.       ELSE IF c > 0 THEN
10.          y ← y + 4
11.      ELSE
12.        y ← y + 3
13.      ENDIF
14.  ENDWHILE
15.  OUTPUT C
```

Python

```
1.   i = 0
2.   y = 1
3.   while y > -1:
4.       i = i + 1
5.       y = y - 2
6.       c = i + y
7.       if c > 3:
8.           y = y - 1
9.       elif c > 0:
10.          y ← y + 4
11.      else:
12.        y ← y + 3
13.
14.
15.  print(c)
```

In pseudocode:

← means assignment: evaluate the expression on the right and store the result in the variable on left.

Example

```
i ← 0
y ← 1
c ← i + y
```

`c` is assigned the sum of `i` and `y` (which is 0 + 1 = 1)

Foundational concepts (5 minutes)

1. What is the difference between pseudocode and a Python program.

2. Define a variable.

3. State all the variables from **Figure 1**.

4. What is a data type?

5. What is the data type of the variables from **Figure 1**? Justify your answer.

6. On which line is an arithmetic operation first used in **Figure 1**.

Program flow control

The pseudocode from **Figure 1** is repeated below.

- Line numbers are included but are not part of the program.

Figure 1

```
1.   i ← 0
2.   y ← 1
3.   WHILE y > -1
4.      i ← i + 1
5.      y ← y – 2
6.      c ← i + y
7.      IF c > 3 THEN
8.         y ← y – 1
9.      ELSE IF c > 0 THEN
10.        y ← y + 4
11.     ELSE
12.       y ← y + 3
13.     ENDIF
14.  ENDWHILE
15.  OUTPUT C
```

1. What is an IF-ELSE-IF structure called **and** what is its purpose.

2. State all the conditions that are being checked in **Figure 1 and** their line numbers.

3. What is the data type of a condition in programming? Explain your answer.

4. A WHILE structure in a program is an example of iteration. What kind of iteration does the WHILE loop in **Figure 1** represent.

5. What kind of iteration is absent from **Figure 1** and how is it usually performed in code.

Code tracing

> ***Code tracing*** *means carefully following a program's logic step by step just like the computer does – to see how the values of variables change as each line runs.*

We trace code to:

- Understand how the program works
- Predict the output
- Spot errors or unexpected behaviour
- Improve our debugging skills

It is like being a detective: you look at the clues (the code) and figure out exactly what is happening in the program over time.

The pseudocode from **Figure 1** is repeated below.

- Line numbers are included but are not part of the program.

```
1.    i ← 0
2.    y ← 1
3.    WHILE y > -1
4.       i ← i + 1
5.       y ← y – 2
6.       c ← i + y
7.       IF c > 3 THEN
8.          y ← y – 1
9.       ELSE IF c > 0 THEN
10.         y ← y + 4
11.      ELSE
12.        y ← y + 3
13.      ENDIF
14.   ENDWHILE
15.   OUTPUT C
```

Complete the trace table for the above program.

You may not need to use all the rows in the table.

i	y	c
0	1	

Skeleton code – Artefact explorer

In this game, the player is an explorer searching for a hidden artefact on a 10x10 grid.

↑ y											
9											
8											
7											
6											
5											
4						**X**					
3											
2											
1											
0	**P**										
	0	1	2	3	4	5	6	7	8	9	→ x

The player starts at position (0,0) on the grid, shown in the above example as **P**.

The player can move in one of the following directions: North, South, East, or West to find the artefact.

The position of the artefact on the grid is randomly generated, shown in the above example as **X** at position (5,4)

After each move, the player is told how many steps away they are from the artefact.

The player must guess the correct direction to move in to find the treasure with as few moves as possible.

- Play the game.

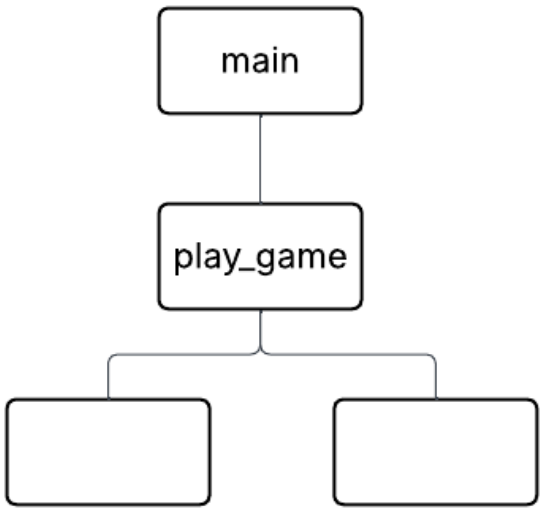
Modular design and decomposition

This program is decomposed into smaller separate functions or subroutines, each handling a specific part of the game. This approach is called **modular design** or **function decomposition**.

This approach helps to make the code easier to read, understand, and maintain - test and debug parts independently, and extend the program later.

Hierarchy chart

> *A **hierarchy chart** is a diagram that shows the breakdown of a program into its main modules or functions and how they relate to each other.*



- The `main` function is at the top, responsible for setting up the game and starting it.
- It calls the `play_game` function, which controls the game loop and player interaction.
- Complete the above **diagram** with the missing functions.

Code analysis

Manhattan distance

```
def get_distance(player_x, player_y, artefact_x, artefact_y):
    return abs(player_x - artefact_x) + abs(player_y - artefact_y)
```

- What does the function `get_distance` calculate?

- What would the function return if `player_x = 2`, `player_y = 3`, `artefact_x = 4`, and `artefact_y = 1`?

- Why might the game use this kind of distance instead of actual diagonal distance?

Player movement within the grid

Here is the part of the program in the `play_game` subroutine that controls the player's movement.

```
if direction == 'N' and player_y < grid_size - 1:
    player_y += 1
elif direction == 'S' and player_y > 0:
    player_y -= 1
elif direction == 'E' and player_x < grid_size - 1:
    player_x += 1
elif direction == 'W' and player_x > 0:
    player_x -= 1
else:
    print("Invalid move. Try again.")
    invalid_move = True
```

Explain the purpose of each of the above conditions?

Programming tasks

Task 1 – Track number of moves

This question refers to the subroutine `play_game`

Currently, the player is not told how many moves they have made in all to find the hidden artefact.

What you need to do

Modify the subroutine `play_game` so that the number of moves the player makes is tracked using a meaningful variable. Initialise the variable appropriately and place in a correct location.

The number of moves should be increased after each valid move.

When the player finds the hidden artefact, the following message is displayed:

`"Congratulations! You found the artefact in xxx moves"` where `xxx` is the total number of moves made.

Guiding questions:

- What type of variable would be suitable for storing a count?
- What would be a meaningful variable name?
- What initial value should you assign to the variable?
- Where in the `play_game` subroutine should you initialise the variable?
- When should the variable be incremented?

- How will you use this variable to display a message when the artefact is found?

Test

Test your changes work by running the Skeleton Program and ensuring that the correct number of moves is displayed.

Task 2 – Limit the number of allowed moves

This question refers to the subroutine `play_game`

The player should be limited to a maximum number of allowed moves to find the hidden artefact. If the player run out of moves, the game ends.

What you need to do

Modify the subroutine `play_game` so it calculates the maximum number of allowed moves as 8% of the total number of cells in the grid in a correct place and stores its value in a meaningful variable.

Display an appropriate message to the player if they have run out of moves.

Guiding questions:

- How can you calculate the total number of cells in a square grid?

  Remember, the size of the grid is already passed as a parameter to the `play_game` subroutine, so you can use it directly.

- How can you calculate 30% of the total cells to set the maximum allowed moves?

- What would be a meaningful name for the variable storing this value?

- Where should this calculation be placed inside the `play_game` subroutine?

- How can we stop the game when the player runs out of moves, and how does this affect our loop condition?

- How will you check if the player has reached the maximum number of allowed moves inside the game loop?

Test

Test your changes work by running the Skeleton Program and ensuring that the correct message is displayed when the user runs out of moves.

Task 3 – Add a hint system

The program is to be extended so it allows the player to get a hint as to which direction they must move to get closer to the hidden artefact.

For example, if the player's current position is `(3,2)` and the artefact is at `(5,4)` then hint returned should be `"NorthEast"`. This is because the `y` position needs to increase from 2 to 4 (so the player must move North) and the `x` position needs to increase from 3 to 5 (so the player must move East).

What you need to do

Create a new subroutine `get_hint` which is passed the player's and the artefact's `x` and `y` position. It should compare the two positions and return a directional hint that guides the player.

Modify the `play_game` subroutine so the input prompt allows the user to enter `H` for a hint (in addition to `N`, `S` `E`, `W`)

When the player enters `H`, the program should call the subroutine `get_hint` using the current positions and display the returned hint. Do not count the hint as a valid move.

Test

Test your changes work by running the Skeleton Program.

Artefact explorer initial code

```
import random

def get_distance(player_x, player_y, artefact_x, artefact_y):
    return abs(player_x - artefact_x) + abs(player_y - artefact_y)

def print_menu():
    print("\n--- Artefact Explorer ---")
    print("Find the hidden artefact in a 10x10 grid.")
    print("Use N, S, E, W to move. Try to find it in as few moves as
possible!")

def play_game(player_x, player_y, artefact_x, artefact_y,grid_size):
    print_menu()
    artefact_found = False

    while not artefact_found:
        direction = input("Enter direction (N/S/E/W): ").upper()

        invalid_move = False

        if direction == 'N' and player_y < grid_size - 1:
            player_y += 1
        elif direction == 'S' and player_y > 0:
            player_y -= 1
        elif direction == 'E' and player_x < grid_size - 1:
            player_x += 1
        elif direction == 'W' and player_x > 0:
            player_x -= 1
        else:
            print("Invalid move. Try again.")
            invalid_move = True

        if not invalid_move:
            distance = get_distance(player_x, player_y, artefact_x,
artefact_y)
            if player_x == artefact_x and player_y == artefact_y:
                artefact_found = True
            else:
                print(f"You're {distance} steps away.")

def main():
    # Game grid size
    grid_size = 10

    # Player starting position
    player_x = 0
    player_y = 0

    # Random location for artefact
    artefact_x = random.randint(0, grid_size - 1)
    artefact_y = random.randint(0, grid_size - 1)

    play_game(player_x, player_y, artefact_x, artefact_y, grid_size)

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

Summer work

Sign up for the following introductory online course to programming using Python.

https://www.futurelearn.com/courses/start-coding-today-an-intro-to-python-programming-for-beginners

- The course must be taken within **2 weeks** with a **3-hour weekly** study. You can start anytime.
- You will receive a **PDF Certificate of Achievement** when you have successfully completed the course.
- **Bring in your digital certificate of achievement to your first Computer Science lesson between the 03 - 05 September 2025.**

IMPORTANT: The course suggests Visual Studio Code to write and edit your Python program. At college we use PyCharm instead. You can download a **PyCharm community edition** for free and install it on your computer: **https://www.jetbrains.com/pycharm/download/**

There are **various online editors** as well, such as

- **https://replit.com**

- **https://editor.raspberrypi.org/en/projects/blank-python-starter**

Sign up for an account so you can save your program.

AQA A-Level Computer Science specification

The course syllabus can be found at:

https://www.aqa.org.uk/subjects/computer-science/a-level/computer-science-7517/specification/subject-content

My Revision Notes