Learning Aims

- Define what is meant by pseudocode
- Learn how and when different data types are used
- Learn the arithmetic, relational and logical operations available in a typical programming language
- Become familiar with basic string-handling operations
- Distinguish between variables and constants
- Learn about how to use selection, sequence and iteration

- **Sequencing** Do one statement after the other
- **Iteration** Do a set of statements multiple times. Iteration is either '**count controlled**' or '**condition controlled**'. Count repeats the section n times, condition waits until a condition has been met before stopping iteration.
- **Branching/Selection** Do a set of instructions based on conditions e.g If ... Else.

Key concepts to remember

Term	What it means	How to do it in Python

Iteration	A loop. There are two loops –
	
	Conditional loops or while loops (Repetition)
	
	for loops (a set number of iterations)
		`number = 0`
		`while number <= 5:`
		`  print("Hello")`
		`  number = number + 1`
		`print("Goodbye")`
		
		`for line in range(4):`
		`  print("hello")`

Variable	Something you can give a value to and then change it at other times in the program.	`name = "Rhiannon" # name is a variable` `number = 56 # number is a variable`

Selection	Where there is a choice point in the program design and an if statement is used to create more than one possible pathway.	`if answer == "Paris":`
		`  print("Correct")`
		`else:`
		`  print("Not correct")`

Input/Output	Getting input from the keyboard or outputting something to the screen.	`name = input("What is your name?")` `print(name)`
Assignment	Where a variable is given a value	`number = 56 # number is assigned the value 56` `name = "Rhiannon" # name is assigned the value Rhiannon`

Key concepts to remember

Selection	Controlling the flow of execution in programs using if and Switch/Case statements
Iteration	Conditional loops or while loops (Repetition) for loops (a set number of iterations)
Condition	Used to control the flow of execution in a program. A condition contains a logical expression. An expression that results in either True or False.
Relational operators	== (equal to) != (not equal to) > (greater than) < (less than) >= (greater than or equal to) <= (less than or equal to) Used in conditions, if x > 10:
Logical operators	AND → True if **both** conditions are True OR → True if **at least one** condition is True NOT → Reverses the Boolean value (True → False, False → True) Used in logic: if (x > 10) AND (y < 5):

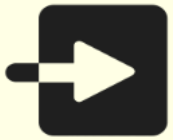
Data Types

- The basic data types which can be used in a high-level programming language are as follows:

Data Type	Description	Example
Boolean	True or false (or any data that only has two possible values)	**TRUE** or **FALSE** **YES** or **NO**
Character	Any single character you see on the keyboard (and more)	'**A**', '**z**', '**8**','**?**'
String	A number of characters	"**Hello World**"
Integer	Whole numbers, positive and negative	**23**, **-45**, **0**
Real/ Float	A number which can contain a fractional part	**15.7**, **-19.25**, **8**

Arithmetic Operators

Operator	Meaning	Example Code	Result
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	7 * 2	14
/	Division (float result)	9 / 2	4.5
//	Floor Division	9 // 2	4
%	Modulus (remainder)	9 % 2	1
**	Exponent (power)	2 ** 3	8



Converting data types (Casting)

When asking for an input the value will always be a string:

```
num = input()
```

To do a calculation the num has to be converted to an integer:

```
calculation = int(num) * 2
```

Calculation will need to be converted back to a string if you print like this:

```
print("Answer" + str(calculation)
```

However not if you do this:

```
print("Answer", calculation)
```

The Round function

You can round this number using a function round.

```
billBetween3 = round(billBetween3,2) #round to 2 decimal places
```

This will return the value 6.67.

String-handling functions

Programming languages have a number of built-in string-handling methods or functions. Some of the common ones in a typical language are:

len(string)	Returns the length of a string
string.find(str)	Determines if str occurs in string. Returns index (the position of the first character in the string) if found, and -1 otherwise. In our pseudocode we will assume that string (1) is the first element of the string, though in Python, for example, the first element is string (0)
ord("a")	returns the integer value of a character (97 in this example)
chr(97)	returns the character represented by an integer ("a" in this example)

To concatenate or join two strings, use the + operator.

name = "Johnny" + "Bates"

Variables and constants

- **A variable is a name/symbol which represents a value in a program**
- ... points to a memory location
- ... and the value be changed (while the program is running)
- ….variable can have no value assigned at design time
- …..you can assign a value to a variable at run-time

- **A constant is also a name/symbol which represents a value in a program**
- ….the value of a constant cannot be changed once the program is running/can only be set at design time
- …..the value of a constant is set when the constant is declared

Variables are identifiers

Variables are **identifiers** (names) given to memory locations whose contents will change during the course of the program.

Identifier

assignment

```
myname = input("Please enter your name: ")
```

Need to know Key Terms

Identifier: the name given to a variable: e.g. score, numberOfLives.

Declare: Some programming languages require variables to be declared first before they can be used. When declaring a variable you give it an **identifier** (and a **data type** for some languages) e.g.
int score

Initialise: To give/**assign** a variable its **first** value which can be changed later on in the program. e.g.
score = 0

Assign: To give or change the value of a variable, using the assignment operator (=) e.g.
numberOfLives = 3

Casting: to convert/change the data type of a variable e.g. in Python using functions such as int(), str(), float().

Increment: To increase the value a variable by a value *score += 1*

Decrement: To decrease the value a variable by a value
timer -= 1

Guidelines could include:

- Start all variable names with a lowercase letter
- Do not use underscores in the middle of variable names
- Use "camelCaps" to separate parts of a variable name – for example, timeInMinutes, maxTemperature
- Do not use overly long names but keep them meaningful – maxTemp is better than maximumTemperature if there is not likely to be any confusion over the meaning of max
- Use all uppercase letters for constants, which are then instantly identifiable
- When defining a class in object-oriented programming, start with an uppercase letter, with the rest of the class name lowercase

Following guidelines such as these will save a lot of time in looking through a program to see whether you called something best_score, Best_Score, bestScore or some other variation.

Operators

- The basic operators which can be used in a high-level programming language are as follows:

Operators	Description	Example
> >=	Greater than/ greater than and equal to	if i > 10
< <=	Less than/ less than and equal to	if i < 10
==	Equal to	flag == True
!=	Not equal to	flag != True
and	Both conditions have to be True	if i > 10 and flag == True
or	Either conditions have to be True	if i > 10 or flag == True
not	condition have to be False	if i > 10 not flag == True

Selection

	Pseudocode	Python	

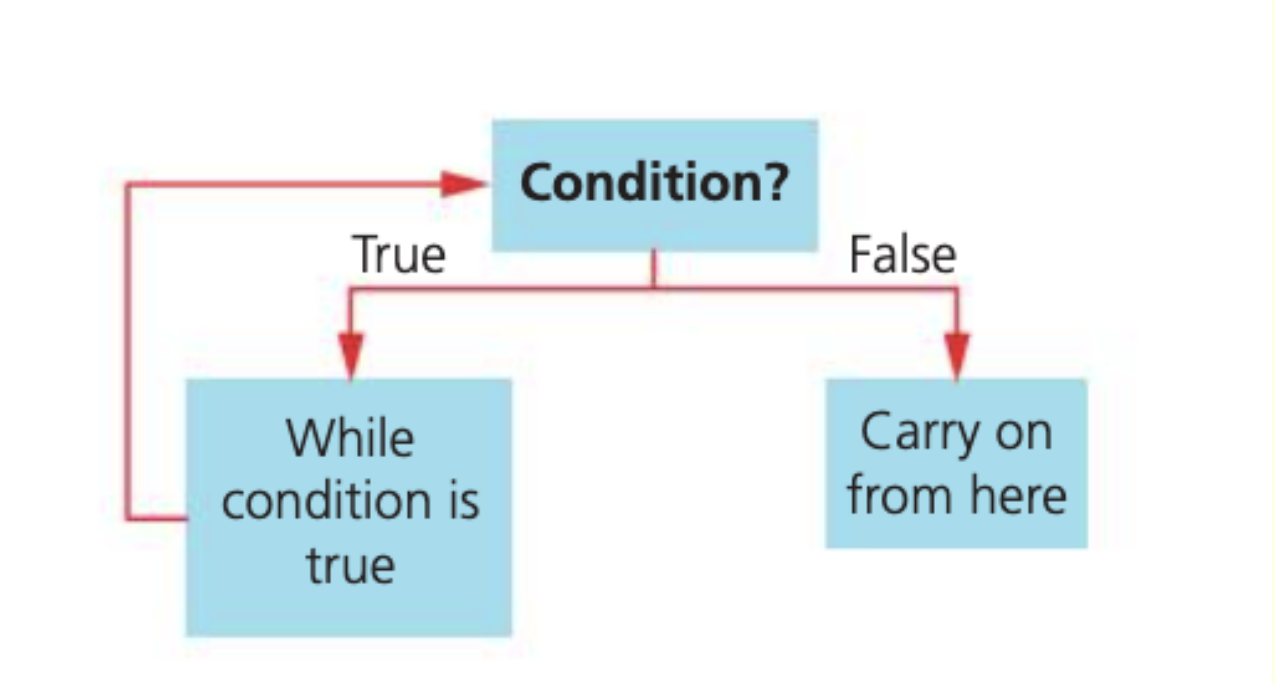
if…else	day = 2
	IF day = 1 THEN
	  print("Monday")
	ELSE
	  print("Not Monday")
	ENDIF
		day = 2
		if day = 1:
		  print("Monday")
		else:
		  print("Not Monday")
			Use **if…else** when there are exactly 2 possibilities.

if…elif…else	day = 2
	
	IF day = 1 THEN
	  print("Monday")
	ELSE IF day = 2 THEN
	  print("Tuesday")
	ELSE IF day = 3 THEN
	  print("Wednesday")
	ELSE
	  print("Invalid day")
	ENDIF
		day = 2
		
		if day = 1:
		  print("Monday")
		elif day = 2:
		  print("Tuesday")
		elif day = 3:
		  print("Wednesday")
		else:
		  print("Invalid day")
			Use **if…elif…else** when you need to check multiple conditions.

Switch case	day = 2
	
	SWITCH day
	  CASE 1:
	  OUTPUT "Monday"
	  CASE 2:
	  OUTPUT "Tuesday"
	  CASE 3:
	  OUTPUT "Wednesday"
	  DEFAULT:
	  OUTPUT "Invalid day"
	ENDSWITCH
		Not supported in python	Use **Switch…case** when there are many discrete choices → cleaner and easier to maintain.

Iteration

- Do a set of statements multiple times.
- Iteration is either '**count controlled**' or '**condition controlled**'.
- Count repeats the section n times,
- Condition waits until a condition has been met before stopping iteration



Iteration

Iteration in high level languages can be done in one of 3 ways:

for Loop
Execute a sequence of statements multiple times
while Loop
Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
do while Loop
Like a while statement, except that it tests the condition at the end of the loop body.

Pseudo coding Loops

Pseudo coding	Python
Count Controlled for i=0 to 7 print("Hello") next i Will print hello 8 times (0-7inclusive).	for i in range(8): print("Hello")
While (Condition Controlled) while answer!="computer" answer=input("What is the password?") endwhile	while answer!="computer": answer=input("What is the password?")
Do until (Condition Controlled) do answer=input("What is the password?") until answer=="computer"	Not supported in python

Flags and counters

- **Flag** → flexible stopping condition (e.g., user input, event).
- **Counter** → fixed number of iterations, controlled by incrementing a variable.

Pattern	Description	Example pseudocode	How it Works

While with a Flag	Uses a boolean variable (flag = True/False) to control when the loop should stop. Often used when the stopping condition is based on user input or an event.	# While loop with a flag
		flag = True
		
		while flag
		  # handle spaces & case
		  answer = input('Type 'exit' to stop: ").strip().lower()
		  if answer == 'exit' then
		  flag = False
		  print("Exiting the loop...")
		  else
		  print("You typed:", answer)
		  endif
		End while
		
		print("Loop finished!")
			1. Loop continues as long as flag is True.
			2. User types "exit"
			3. flag becomes False
			4. loop ends.

While with a Counter	Uses a numeric counter to repeat a fixed number of times. Often used instead of a for loop when manual control is needed.	#While loop with a counter
		count = 0
		limit = 5 # makes it easy to change how many times the loop runs
		
		while count < limit
		  print("Count is:", count)
		  count += 1
		End while
		print("Loop finished!")
			1. Loop runs while count < 5.
			2. Each iteration adds 1 to count.
			3. When count reaches 5, condition is False
			4. loop ends.