OCR A Level Computer Science

Your notes

6.4 Thinking Logically

Contents

Decisions in Computational Thinking

Your notes

Identify the Points in a Solution where a Decision has to be Taken

- Most languages use **structured programming techniques** including:
  - **Sequence**: one statement after another
  - **Selection**: Decision making, if/then/else, switch/case
  - **Iteration**: Loops, for/while/do while/do until
- The purpose of these techniques is to **aid the readability**, **understanding** and **maintainability** of code
- **Python** is an example of a **block-structured language**, which uses only these three constructs to control the flow of execution and data in a program. Each block of code should have a single entry and exit point and ideally minimise breaking out of iterative blocks. This is to prevent unintended consequences relating to the flow of control of a program such as entering or leaving a subroutine early or branching unintentionally
- When designing algorithms, it is always best to **plan the algorithm** using **flowcharts**, **pseudocode** or s**tructured english** before coding in a language
- **Languages** have specific **syntax**, **constructs** and **idiosyncrasies** that differ between other languages. This makes it challenging to create a solution that can immediately be implemented in another language
- When determining the points in a solution where decisions are made, **flowcharts are useful** as they **visually** show the flow of control in a program. Decisions are clear and easy to follow, however flowcharts are **time consuming to create**
- **Pseudocode** more **accurately** mimics the constructs of programming languages without the problem of syntax. As pseudocode has no syntax, any way of expressing an algorithm is acceptable as long as the pseudocode **meaning is clear**
- **Structured English** can be an alternative to pseudocode but is usually more **verbose** and **imprecise**

How do you Identify the Points in a Solution where a Decision has to be Taken?

- Most errors in a program occur when evaluating a **Boolean condition**, whether in a **sequence** as part of a statement, **iteration** or **selection**. It is therefore important to be careful when creating **Boolean** conditions, especially long, complex conditions involving multiple clauses

- **Decisions** in programs usually occur in two situations, an **if statement/select case** or a **loop**, usually a **while loop**
- **Selection**, also known as **branching**, involves directing the flow of control of a program, dependent on a **Boolean** condition or set of conditions
- **Iteration** involves **repeating** a sequence of instructions based on a stopping **Boolean** condition


Your notes

Determine how Decisions Affect Flow through a Program

- Decisions **affect** the **flow of control** of a program
- Decisions are **Boolean conditions** encountered in **selection** structures such as **if/then/else** statements and **iteration** structures such as **while loops**
- **If** statements and **switch/case** statements can consist of many decision points as illustrated below. Each decision point directs the program through different statements

if/then/else example

```
if today == "Monday" then
    print("Eugh! Monday again!")
elseif today == "Tuesday" then
    print("Tuesday, one day closer the weekend!")
elseif today == "Wednesday" then
    print("Half way there!")
elseif today == "Thursday" then
    print("One more day to go!")
elseif today == "Friday" then
    print("I can't believe its Friday!")
elseif today == "Saturday" then
    print("Woo! Its Saturday!")
elseif today == "Sunday" then
    print("Aww, its Monday tomorrow!")
```

```
else
    print("That's not a day!")
endif
```

switch/case statement example

```
switch entry:
    case "Monday":
        print("Eugh! Monday again!")
    case "Tuesday"
        print("Tuesday, one day closer the weekend!")
    case "Wednesday":
        print("Half way there!")
    case "Thursday":
        print("One more day to go!")
    case  "Friday":
        print("I can't believe its Friday!")
    case "Saturday":
        print("Woo! Its Saturday!")
    case "Sunday":
        print("Aww, its Monday tomorrow!")
    default:
        print("That's not a day!")
endswitch
```

- **Iteration** and **selection** statements can be **nested**, leading to a structure as shown below

```
Count ← 0
REPEAT
```

Your notes

```
  INPUT Score[Count]
  IF Score[Count] >= 70 THEN
    Grade[Count] ← "A"
  ELSE
    IF Score[Count] >= 60 THEN
      Grade[Count] ← "B"
    ELSE
      IF Score[Count] >= 50 THEN
        Grade[Count] ← "C"
      ELSE
        IF Score[Count] >= 40 THEN
          Grade[Count] ← "D"
        ELSE
          IF Score[Count] >= 30 THEN
            Grade[Count] ← "E"
          ELSE
            Grade[Count] ← "F"
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
Count ← Count + 1
UNTIL Count = 30
```

- Each **if statement** contains another **if statement** which affects the flow of the program. This is however functionally identical to the days of the week program shown above

Your notes

Logical Conditions in Computational Thinking


Your notes

Determine the Logical Conditions that Affect the Outcome of a Decision

- Most errors in a program occur when evaluating a **Boolean condition**, whether in a **sequence** as part of a statement, **iteration** or **selection**. It is therefore important to be careful when creating **Boolean** conditions, especially long, complex conditions involving multiple clauses

What are Boolean Conditions?

- **Boolean** conditions use combinations of several operators:
  - **==** (equal to)
  - **>** (greater than)
  - **>=** (greater than or equal to)
  - **<** (less than)
  - **<=** (less than or equal to)
  - **!=** (not equal to)
  - **AND** (both conditions must be true)
  - **OR** (either or both conditions must be true)
  - **NOT** (the condition is not true)
  - **XOR** (only one or the other condition must be true)
- An example of a condition could be a teenager wanting to see a 15 rated movie at a cinema that costs £8. If the teenager isn't at least 15 years old and doesn't have enough money then they cannot watch the movie

```
if age >= 15 and money >= 8 then

entry = True

endif
```

- Another example could be checking whether a list contains any numbers and if so, removing them. In order to do this, each item must be checked one by one, checked to see if it is numeric then remove it if so

```
i = 0
```

```
valid_element = True
while len(my_list) > 0:
  valid_element = True
  element = str(my_list[i])
  for index in list_item
    if list_item[index] in [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] then
      valid_element = False
      break
    endif
  next index
  if valid_element == False then
    my_list.pop(i)
  endif
endwhile
```

Your notes

- The above algorithm is a **pseudocode solution** to **removing numbers from a list of strings**
- The **decision** points are:
  - **While loop**: the list has to have at least one element in order for the while loop to run
  - **For loop**: all elements in the list "list_item" are iterated over one at a time. The stopping condition is reaching the end of the list
  - **First if statement**: each character in the list "list_item" is checked. If it is a numeric digit then the string contains a number and therefore must be removed. Be aware that in this implementation, a string is considered numeric if it contains even a single digit. Valid_element is set to false and the program breaks out of the for loop
    - Be careful using "**break**" instructions. They are powerful and help control the flow of control but can be misused and cause unintended problems
    - **Second if statement**: if the item in "my_list" contains a digit, it is removed from the front of the list