Adam Suttle 12A 13.11.16

Miss Berry Topic Revision November 2016

Topic 1 – Programming constructs

There are 3 main types of programming constructs that are used within source code of programs. These will affect the way in which statements are executed. Early programmes were developed on an `ad hoc' basis, with no particular rules on how they were laid out. Ad hoc is Latin for (for this), meaning create do done for a particular necessary purpose.

Early programs often used GOTO statements which allowed transfer of control form one point of the program to another. This made programmes harder to follow, maintain and debug. Over time 3 new and more popular techniques became adopted:

1. **Sequence:** This simply means that all the instructions of code are executed one after the other in the consecutive order that they appear in.
2. **Selection:** A condition is used to determine which of the statements (if any) will be executed and a s a result some instructions may not be executed.
3. **Iteration:** A group of instructions are repeated for a given number of times or until a condition is met.

Iteration can involve both condition controlled loops and count controlled loops.

A count controlled loop will repeat a set of instructions for a known/set number of times. These are useful when we know how many times we want the code to reiterate.

NOTE: in python when we have a range in a count loop e.g. for i in range (0, 7) . The maximum will mean up to but not including. However, in pseudocode in exams, the max value is up to including. i.e. 0 to 7 in python loops 7 times. 0 to 7 in pseudocode will loop 8 times.

A condition controlled loop will repeat lines of instructions until a condition is met. This can be caused by a change in entered data, random values, calculated results, e.c.t.

Pseudocode formats

```
for I = 0 to 7:
        print("Hello")
next i
```

```
while answer != 1234
        answer = int(input("What is the password"))
end while
```

```
repeat
        Y -= 1  #This is the same as Y = Y – 1 # For addition we can
use Y += 1 (Y=Y+1)
until Y < 0
```

Adam Suttle 12A 13.11.16

```
input BMI

if BMI < 18.5:
        print("Underweight")
if (BMI >= 18.5) AND (BMI <= 25):
        print("Average")
if (BMI > 25) AND (BMI < 30):
        print("Overweight")
else:
        print("Obese")
end if
```

```
input BMI

SELECT case BMI:

    Case < 18.5:
        print("Underweight")
    Case >= 18.5 AND <= 25:
        print("Average")
    Case > 25 AND < 30:
        print("Overweight")
    Case else:
        print("Obese")

END SELECT
```

```
input BMI

Switch BMI:

    Case < 18.5:
        print("Underweight")
    Case >= 18.5 AND <= 25:
        print("Average")
    Case > 25 AND < 30:
        print("Overweight")
    default:
        print("Obese")

End switch
```

SELECT is usually preferred to using multiple statements because it is more concise so:

- **It is easier to read, making source code neater and clearer to understand.**
- **They avoid numerous repeats of similar conditions (multiple branches), <u>because all of the alternatives depend on one variable.</u>**

Topic 2 – Programming errors

There are different types of errors, or bugs, which can prevent a program from working in the way they should. They are categorised as three different types: runtime errors, syntax errors and logical (semantic) errors.

Runtime errors:

- Runtime errors are errors which cause the program or computer system to crash or stop during execution, even when it appears the source code is in working order. This can occur if the computer runs out of memory, e.g. When instructions are written in the wrong order or if a data type does not match what is expected. Runtime errors are identified when the program is being compiled. The compiler will cause the program to output an error that can be used in debugging.

Syntax errors:

- These are errors that occur due to conflicts in the language rules of the source code. Translators can only execute a program if it is syntactically correct. Common syntax errors include spelling mistakes, incorrect use of punctuation, use of capital letters, undefined variables.

Logical (Semantic) errors:

- These are errors in the source code that results in the program producing a different result than what is expected. It can be translated and executed without any errors being reported. The problem can be identified during testing by noting an unwanted output.

Examples can be infinite loops, wrong variable types (e.g. adding two strings rather than integers).

Adam Suttle 12A 13.11.16

Common errors

Indentation = leaving a defined amount of space from the beginning of the statement so that it is shifted to the right.

This is done to statements within a block. This makes code more clear to see where blocks (e.g. sub-routines) start and end. It also tells the program how to interpret the statements, they must be executed within a block/ sub-routine.

- A lack of indentation could cause both logical errors or syntax errors (if code is ran in the incorrect routine or if the programme can't interpret the code, respectively).
- It is very important to format an input to lowercase or uppercase depending on what is desired when comparing it in a selection/branched instruction. This is because programme languages are case sensitive. A uppercase character will have a different character code to a lower case character. This can result in a logical error in the output.
- It is also important to choose the correct data type for variables. If this is not done then logical errors can occur when values are operated in such a way that the result produced is not the desired one.
- When we have a statement such as If GasBill AND ElectricBill > 10 Then
  There is a logical error because `AND' is used to join two expressions yet there is no expression for the variable GasBill as it is not being compared. It is not checking if both values are greater than ten but instead checking if their sum is.

Topic 3 – Programming variables

A variable = is a place in the memory dedicated to storing a value which has the ability to change, throughout the execution of the program, depending on certain conditions or information passed into the system.

A constant = a place in the memory dedicated to storing a value which will not change throughout the execution of the program (e.g. VAT = 20%) . It is fixed and assigned by the program.

Scope = relates to the section of code where the variable is defined. Variables can be local or global.

Variables are used to identify storage locations within the memory. When a variable is declared, it is given a specific name that relates to a storage location allocated in the RAM. This allows programmers to access the value held within a variable from the memory address efficiently all thorough the program. Since the memory is volatile, when the program is ended or the computer shut down. All the data is cleared. Next time the program is run the variable may be assigned a different memory location but it will always be referred to by its variable name. This makes it possible to always identify a memory location for the data that needs to be stored by the program. The value of the variable is not consistent and may change through execution as information is passed in or conditions are met.

Constants work in the same way but their value is always fixed when assigned by the source code. Execution will not alter the contents of the memory location holding the constant.

Data types

Variables when being declared will have a data type. This is used to define the type of data that the variable will hold within the memory. Different data types will have different syntax rules on how the data can be operated on.

Adam Suttle 12A 13.11.16

1. **String** – data type used to store characters (including upper/lower case letters, alpha numeric data (e.g. post codes – KT24 5JR), symbols $%, long numbers that don't require mathematical operations (phone numbers 01372 45****).

String manipulation, string data types can be used in different ways:

a) Substrings, an extracted part of a large string. Strings can be sliced to remove certain parts (e.g. remove the first 3 characters).
b) String functions, operations can be done on strings to alter them. Such as Length, which finds the length of a string returning an integer value; UPPERCASE/LOWERCASE changing the case of the characters in the string; Trim, which will remove any white space (spaces or other given character from the string.
c) Concatenation is the term used to describe the process of joining two strings together. We tend to use "&" in pseudocode as it makes it clear we are not adding (+) but instead concatenating.

2. **Integer** – data type used to store whole numbers, including negatives. It can be treated mathematically with operations to change its value.

3. **Real** (**float** in python) is a data type used when floating point numbers need to be stored (possible of having a decimal result). It is possible to perform a range of mathematical operations on them.

4. **Boolean** is a data type used to store data that can only exist as one of two values (e.g. True or False). It only has two states: 1 (True) & 0 (False).

Declarations & Assignments:

- **Declarations** tell the program the name of constants/variables and give their data type and determine the memory location of the data.
- **Assignments**: Is the act of assigning a value to a variable. Assignments can occur through the execution of the program due to a line of code that alters the variable's previous value.

Identifiers

These give variables (the storage locations in memory) a name that can be referenced throughout the program. They can be made unique in two different ways:

1) Using unique names in their declaration
2) Utilising different scopes within the program. These rules determine where an identifier is defined. Local variables/constants are only defines by their identifier within the function they are created in. Conversely, the global variables/constants given an identifier are defined all through the program.

The scope of variables

The scope of a global variable is the entire program. The scope of a local variable is the sub-program where it has been declared.

- **Local variables** are variables that are created within, and will only operate in a section of code (sub-routine/function/procedure/class). They are created when the routine is called and destroyed when it ends.

…..Variables are declared within a function when they are only needed for use within that routine.

- **Global variables** are variables that are created outside of sub-routines, visible and so accessible to any part of the code within the program. They are created when the program starts and destroyed when it ends.

…..Variables are to be defined globally only if they are required to be accessed all through the source code and do not conflict with any identical identifiers.

Why do we avoid using global variables?

1. Global variables make it difficult to integrate modules
2. They increase the complexity of a program as it can be hard to follow the value within constructs as global variables can be accessed and altered by all parts of the program.
3. They may cause conflicts with identifiers written in other modules and sub constructs and so be changed inadvertently when the program is complex (name space pollution).
4. Limiting the scope of elements by using local variables makes functions reusable
5. Global variable must be accessed by every sub-program or module at all times. This means it is always present in RAM during execution. This reduces efficiency creating more demand on system resources.

Solutions

1. Programs can be safely divided into components where the identifiers within are only specific to that division of source code (modules). This prevents conflicts between different parts of programs/other programs being executed ; the same names for identifiers can be used in different places to refer to different things. These types of identifiers have a module scope: only being defined throughout a single module or file.
2. Using local variables for data that only needs to be operated on within sub-constructs will limit the scope of variables. This prevents name space pollution and prevents conflicts outside the construct it is defined in.

3. **Parameter passing** – This is when a copy of a variable is used to describe the data held within it called an argument. This is passed as the input into a procedure/function meaning that only a copy of the original variable is acted upon. There are no unforeseen effects that occur in other modules.

Passing by value Vs passing by reference

Parameter passing by reference:

When a programmer passes a parameter by reference. A reference of the memory location in the RAM of the original value of the variable is given. Any alterations made to this during execution will apply to the original value of the variable used in the main program.

Parameter passing by value

When a programmer passes a parameter by value. There is a copy made of the original variable which stores a temporary replica of the original value. When acted upon, alterations only occur to the value within the RAM location of this copy variable. The value of the original variable of the main program is unaffected.

Disadvantage - Increased demand on memory resources (RAM), less efficient because a second copy is required to be stored in the memory.

Advantage - The value of the original variable is protected from accidental or unnecessary change.

Topic 4 – Programming conventions

Why do we need conventions when coding (strict rules programmers must abide to, especially when working as a team creating different modules of code)?

- Conventions allow code to by written to the same standards and format. This makes it easier for multiple team members/programmers to follow and understand.
- Keeping code neatly written with spaces and documentation will make the source code more readable and also easier to understand and interpret saving time.
- Splitting code up into modules, procedure, functions will not only limit name space pollution and reduce conflicts of global variables sharing identifiers, but also make the source code on the whole more readable and manageable to write in smaller sections.

These all make the code easier to maintain and debug.

- Writing identifiers to a standard convention and giving them appropriate names will make them easier to understand while avoiding duplication.
- Variables should not be outside the scope of a sub-program (construct) to limit name space pollution, increase efficiency and make debugging across multiple modules more efficient.
- Using functions/procedures makes blocks of code reusable so avoids unnecessary duplication.
- Hardware specific code should be avoided as this limits the compatibility of the program to function on different computer systems with such a diverse range of hardware.

- Functions should not be longer than one page of code since this makes them more complex. Multiple instructions can mean it is difficult to follow the precise state and value of variables and track bugs.
- Functions should have only one entry point to reduce bugs and make it easier to see where a function is being called. This aids debugging as the programmer can follow the state/values of parameters passing to the functions efficiently.

Python conventions for general programming:

- When defining a procedure use names such as proc_name to avoid unambiguity between a procedure and function.
- Likewise functions should be defined as func_name to identify this.
- Identifiers should have meaningful names (give insight to what they are and avoids duplicates)
- Constants are generally defined using BLOCK CAPITAL identifiers.
- Variables should not being with numbers. They can only be identified using letters, numbers and underscores.
- When identifying a global variable be explicit e.g. globalNAME = Adam
- Variables cannot use key word of the program language as this will cause syntax errors.
- If a language is case sensitive then you should be particularly careful with variable names.
- Never use characters like l (lower case `el') or O (uppercase `oh') or an I (upper case i) as single character names. They can be confused with similar characters making them hard to distinguish in different fonts. (like 1 el, I and roman numerals or zero)

  - Class names – should by CapWords (CamelCase) by convention. This is capitalising first letter of each word in identifier.
  - Constants can use uppercase and underscores only
  - General variables should only use lowercase and underscores and be kept short
  - Function names should use the lowercase words separated by underscores.

Camel case convention is called this because the first letter of every word becomes capitalised, exhibiting a bumpy appearance like a camel's back.

Topic 5 – Procedures, Functions & Modules

A sub-routine = a named block of code that carries out a specific task.
For example: functions & procedures.

Why are subroutines important:

When a specific set of instructions will be repeated a number of times at different points in a program, to avoid unnecessary repetition they can be put into a spate sub-routine. This named block of code can be called from any point in the program to carry out the task. This also makes code easier to maintain as it is more readable and clear to follow.

Adam Suttle 12A 13.11.16

A function = A named sub-routine of code that carries out a self-contained set of instructions to return a value to the main program. It can be accessed from different parts of the program when called, arguments may be inputted when parameters are passed into the.

A Procedure = A named sub-section of code that carries out a self-contained set of instructions but does not return a value to main program. It can be called from different parts of the program and display an output to the user.

The returned data type of a function must be the same as the variable that it is stored in.

Functions can be both user defined or part of a program language. E.g. print(" ", variable) or int(…) or input(…) are all believe it or not - functions!

Sub-routines are used because:

- It saves time in coding as the same code does not have to be re-written. The unnecessary repetition is removed making code more readable and clear.
- It makes programmes more efficient requiring less memory/storage space.
- It divides code up into more manageable and readable sections that are easy to debug.
- Variables defined within them are local scope and so avoid the sue of global variables, preventing name space pollution.
- Parameters can be passed by value into a procedure/function so only a copy of the original variable is altered, protecting the value in the main program and preventing inadvertently editing values.

A parameter = a special king of variable **passed into a sub-routine when it is called**. It is a copy of a variable containing a piece of data called an argument. Only a copy of the original variable is acted upon.

A Module = a subsection or construct within a larger program, where the program is divided into sections of **related functionality**.

In reality, large programs are written with many modules that have been created in other projects possibly by different programmers. This reiterates the need for conventions.

Adam Suttle 12A 13.11.16

Why is a modular approach advantageous?

1. It allows large workloads to be divided between a team (manageable tasks)
2. It allows for work place to take place in parallel (multiple programmers) so save time
3. It breaks the program up into parts so that programmers can focus on their expertise.
4. It breaks difficult problems up into smaller areas
5. It means that sections of the program are reusable in future projects
6. It can make large programmes much easier to maintain, test, read and debug.
7. Each sub-routine or module can be tested before integration with other modules.
8. Each team member only requires to know the input values of their sub-routine and the expected values outputted.