Programming Structure

Programming Structure or Structured Programming is a concept with the purpose of improving the quality, clarity and time spent developing code of a computer language. This concept has a number of different elements associated to it that are common throughout most programming languages. These common elements improve the flow and order of code. It avoids developers producing "spaghetti code" which can be messy, difficult to understand and difficult to maintain.

Control Structure:

Control Structure is defined as:

"a block of programming that **analyzes variables** and **chooses a direction in which to go based on given parameters**. The term *flow control* details the direction the program takes (which way program control "flows"). Hence it is the **basic decision-making process in computing**; flow control **determines how a computer will respond when given certain conditions and parameters**."

When a program is running, the code is being read by the computer line by line (from **top to bottom**, and for the most part **left to right**), just like you would read a book. This is known as the "**code flow**", now as the code is being read from top to bottom, it may hit a point where it **needs to make a decision**, this decision could make the code:

- jump to a completely different part of the program,
- or it could make it re-run a certain piece again,
- or just plain skip a bunch of code.

You could think of this process like if you were to read a choose your own adventure book, you get to page 4 of the book, and it says "if you want to do *X*, turn to page 14, if you want to do *Y*, turn to page 5". That **decision** that must be made by the reader is the same **decision** that the computer **program must make**, only the computer program has a **strict set of rules to decide which direction to go.**

So, these decisions that must be made, that will in turn **affect the flow of code**, are known as a **control structure**!

Examples in programming of these include:

- ***SELECTION***
  if else statements which either run or skip a block of code depending on whether a condition is met (**if** the person is over 18 years old, *sell them alcohol*, **else** *refuse them alcohol*).

```
if( personAge > 18 ) {
        sellAlcohol();
}
else {
        doNotSellAlcohol();
}
```

- ***ITERATION***
  for and **while** loops are examples of code that will continuously repeat a block of code for as long as a condition is met (**while** John is hungry, *feed John food*).

```
while( john == hungry ) {
        feedJohnFood();
        hungry -= 1;
}
```

- ***SEQUENCE***
  The way in which code is order from top to bottom, left to right and read in sequence.

Blocks:

A **code block** or simply a **block** is a section of code which is grouped together. As with most languages, a block of code will be enclosed by curly brackets **{ }** and will normally have a statement or declaration in front of it. Blocks can be nested inside other blocks and are fundamental to programming structure.

```
while( john == hungry ) {
        feedJohnFood();
        hungry -= 1;
}
```

Subroutine:

Subroutines are blocks of code which can be referred to by name or a single name. This allows a developer to group similar lines of code together for a singular purpose perhaps and then can invoke or execute this code whenever they want by referring to it by name. In programming, depending on the language you are using, these may be called: **functions**, **methods** or **procedures**.

```
function sellAlcohol() {
    // tell customer price
    // take customer's money
    // give customer change if there is any
    // give customer bag of booze
}
```

Variables

Variables are paramount to any program. They are the essential building material needed to put together effective code.

Wikipedia describes variables in computer programming as:

"a **storage location** and an associated **symbolic name** which contains a **value** which is some known or unknown quantity or information."

In simpler terms, a variable holds onto some sort of information (a **value**) which we can use or alter later on. We will give this variable a **name** so whenever we want to retrieve the information in the variable, all we have to do is refer to its name.

Let's use another real world example. Let's create a variable which will refer to the amount of money that you have in your wallet. We'll give this variable the name **myMoney** and since you have $20 in your wallet we'll give it a value of **20** (**myMoney = 20;**).

If you were to buy a coffee at a cost of $3 dollars then we should update **myMoney** to reflect the change in its value. We could this by writing something like

```
myMoney = 17;
```

or by doing something like this

```
myMoney = myMoney - 3;
```

Using variables like this allows us to keep track of things happening in our code in a much easier fashion. If we wanted to change the value of the amount of money in your wallet, all we have to do is assign the variable name a new value as demonstrated above. That can be done in a single line of code which will affect the rest of the program. If we didn't do this we may have had to alter numerous numbers scattered throughout the code which is very time consuming and not adaptable.

Variables are **declared** when they are first created.

Variables are **assigned** values when they are given their value. This usually occurs with the equals (**=**) sign.

Note that to declare a variable you do not necessarily have to assign it a value. That can happen later in the code.

Variable Types, Data Types

Variables have a particular type which refers to the type information or data that they are storing. As stated in the previous lecture, some languages require you to declare the data type of a variable when declaring the variable itself. These are strongly typed languages such as Java and C++. Other languages don't require the developer to explicitly state the variable type of the variable being declared. These are weakly typed languages such as JavaScript and PHP.

Regardless of whether a language is strongly or weakly typed, it is still very important to know what type of variable you are dealing with. For instance you can't add the number 57 to a string that is "James Finn" and expect to get a normal result.

A computer does not know the difference between "5789" and "My name is Kevin". Since all programming deals with **input**/data and **processes** it to create some useful **output**, we need to be able to classify and sort different kinds of data. If we don't classify or understand data accurately, it becomes difficult to process.

There are 7 main data types/variable types that you need to become familiar with.

Data Type	Explanation	Example
Boolean	True or False.	true false
Integer	Any WHOLE number which can be positive or negative.	10, 2342, -23, 999089, -1, 0
Float/Double	A number which has a decimal point. This can be positive or negative.	10.5, 2342.99, -23.6, -3.14
String	A group or string of text or characters.	“James”, “cat”, “Hello World”, “99”
Char	A single character, letter, number, symbol.	abcdeABCDE12345@#$&
Array	A collection of a series of data organized together in a single unit or group.	[“apple”, “orange”, “banana”]
Object	An instance of a class that can have variables, functions and data structures. Think of it as a more complicated array.	

General Purpose Questions to Ask Yourself Before Programming a Project

1. What is the goal/purpose of the program?
2. What are the programming requirements?
    - *Functions, data, processes, actions, essential components of the program.*
3. What are the constraints/limits?
    - *Timeline, budget, technology.*
4. What are the inputs/outputs?
5. How will the inputs be processed?
6. What can go wrong?
    - *Spot potential bugs that may arise.*