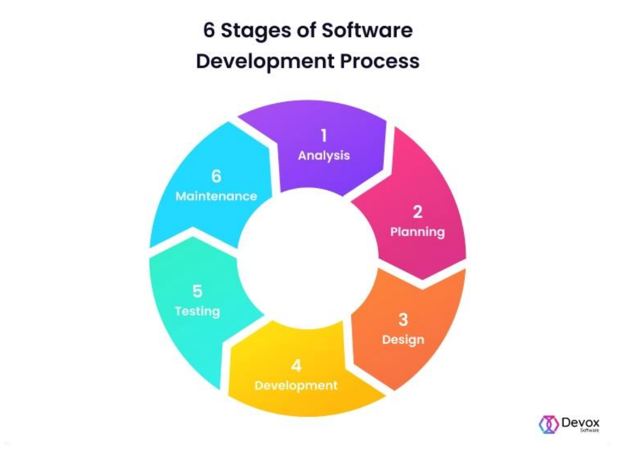
Design Workshop

Assignment 2

Unit 6

George Hotten

Software Development Lifecycle



Analysis – Understanding the scope

You must consider what the program will do and what it won't, what will be included and what will be left out. Depending on resources, the features should be prioritized and the most important should be put into the first version. Other features can be included at a later stage.

Planning – Requirement gathering and specification

You should get the opinion from the users interested in using the program. You should ascertain what they want it to do? However, some users may not be able to give sufficient detail or will use terms that are related to their business that the programmer will not be able to understand. The type of questions that should be asked are: what is the primary aim? How does the current program work? What other programs must it interact with?

Design – Designing the program

Before any programming occurs, the interface should be designed to ensure the users are happy with it. The design should include the structure of the program and how all the different procedures will relate to each other. Ensure the design is detailed and shows how the data should be stored within the program, including file structures and variable names.

Development – Coding the program

This is the stage in the lifecycle where the coding of the program begins. This usually involves multiple programmers each working on a different part of the program.

Testing – does it work?

After the programming has finished, each part of the program must be tested. This involves ensuring that all the aspects of the program are working as expected. This is to ensure there are no bugs and the program works to a high quality. You should also ensure the program has met the original specifications of the project.

Maintenance – Keeping the program up to date

The program should be constantly maintained and updated to ensure the users get the best possible experience. It is natural that some bugs may slip through testing, and when they are discovered, the programmers should fix it and push an update as soon as possible. There may also be other improvements, optimizations and additions to be added to the program, which is where the lifecycle will restart.

Software Structures

Functions

A function is a set section of code that returns a value. This could be built into the programming language to get data such as the date and time, or the programmer can add their own. These can be used to achieve any task that has an output.

Procedures

A procedure is a function that doesn't return any value, which is often referred to as 'void' in many programming languages.

Classes

Classes are templates that defines an object's attributes and methods. Attributes are the variables in the class, whilst methods are the functions and procedures. The class is where all the code for the object occurs.

Objects

Objects can be compared to entities: real-world objects within a system. An object is the instance of the class, which is the template for the object that holds all its attributes and methods.

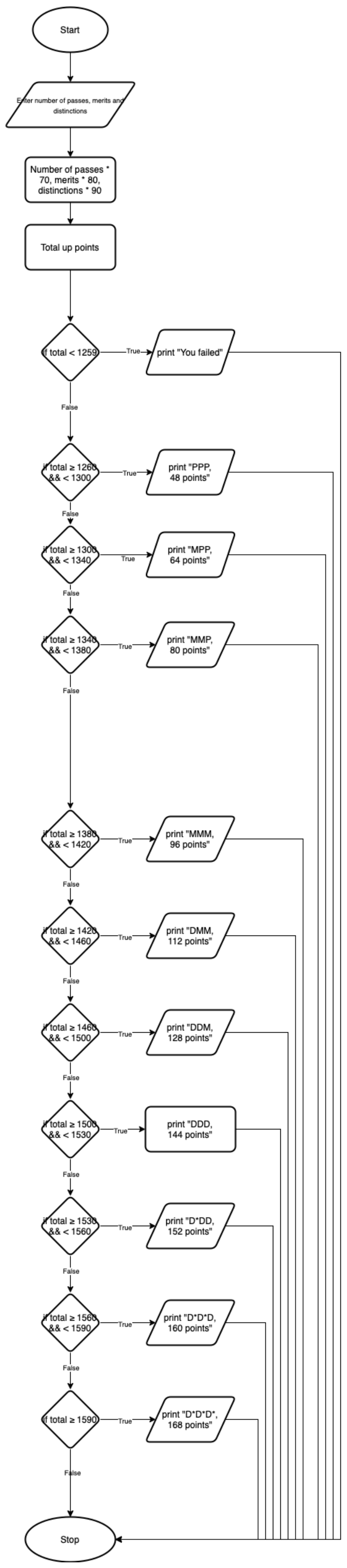
Data abstraction

Data abstraction is the taking of data and removing all unnecessary details. This is usually done for data protection or simplicity. For example, a cleaner doesn't need to be able to access student profiles.

Pre-defined code

Pre-defined code is code that is already built into the system. For example, built in libraries such as DateTime from Java. As they are already defined, they cannot be used as variables as they already have a value or a function.

Grade Calculator Flowchart


Start
Enter number of passes, merits and distinctions
Number of passes * 70, merits * 80, distinctions * 90
Total up points
if total < 1259
True
print "You failed"
False
if total ≥ 1260 && < 1300
True
print "PPP, 48 points"
False
if total ≥ 1300 && < 1340
True
print "MPP, 64 points"
False
if total ≥ 1340 && < 1380
True
print "MMP, 80 points"
False
if total ≥ 1380 && < 1420
True
print "MMM, 96 points"
False
if total ≥ 1420 && < 1460
True
print "DMM, 112 points"
False
if total ≥ 1460 && < 1500
True
print "DDM, 128 points"
False
if total ≥ 1500 && < 1530
True
print "DDD, 144 points"
False
if total ≥ 1530 && < 1560
True
print "D*DD, 152 points"
False
if total ≥ 1560 && < 1590
True
print "D*D*D, 160 points"
False
if total ≥ 1590
True
print "D*D*D*, 168 points"
False
Stop

Flowchart to Pseudocode

```
# Variables
# This asks for the user's input of their passes, merits and distinctions
passes = input("Enter the number of passes: ")
merits = input("Enter the number of merits: ")
distinctions = input("Enter the number of distinctions: ")

# Once inputted, the code multiplies the Ps, Ms and Ds by their appropriate points and adds them up.
total = int((passes * 70) + (merits * 80) + (distinctions * 90))  # integer data type

# Then by comparing the total to the grade boundaries, and then outputting their grade and UCAS points.
if total < 1259:
    print("You failed")
elif 1260 <= total < 1300:
    print("PPP, 48 points")
elif 1300 <= total < 1340:
    print("MPP, 64 points")
elif 1340 <= total < 1380:
    print("MMP, 80 points")
elif 1380 <= total < 1420:
    print("MMM, 96 points")
elif 1420 <= total < 1460:
    print("DMM, 112 points")
elif 1460 <= total < 1500:
    print("DDM, 128 points")
elif 1500 <= total < 1530:
    print("DDD, 144 points")
elif 1530 <= total < 1560:
    print("D*DD, 152 points")
elif 1560 <= total < 1590:
    print("D*D*D, 160 points")
else:
    print("D*D*D*, 168 points")
```

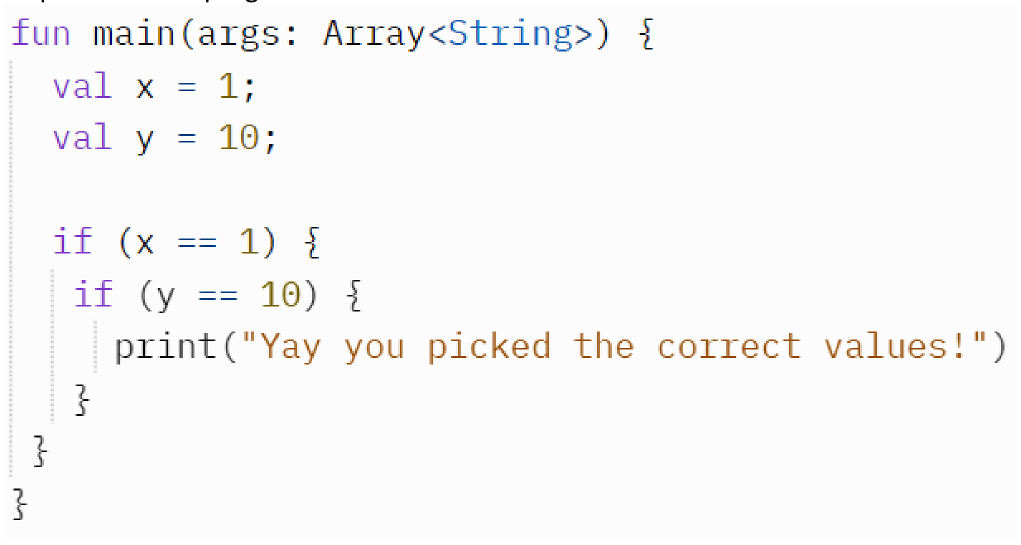
Data type or structure to be used	Variable name	Reason for choice
Integer	passes	This is the number of passes the user has inputted. This is of the data type integer as it is a whole number.
Integer	merits	This is the number of merits the user has inputted. This is of the data type integer as it is a whole number.
Integer	distinctions	This is the number of distinctions the user has inputted. This is of the data type integer as it is a whole number.
Integer	total	This is the BTEC points that are calculated by running `(passes * 70) + (merits * 80) + (distinctions * 9)`
Process		This is the calculation that is done to calculate the total variable as seen above.
If-Else		This is the chain that runs to determine the user’s grade by comparing their BTEC points to the criteria

Code Readability

Indentation

Indentation is used to help programmers read code and understand how it flows. This is often done by adding spaces and tabs in front of code blocks (such as If-Else). This allows programmers to quickly see where the block ends and what code is inside it. This is especially useful in nested If statements.

As you can see from the following code sample, I have intended the code inside the If block to help show other programmers what code is executed in that block.

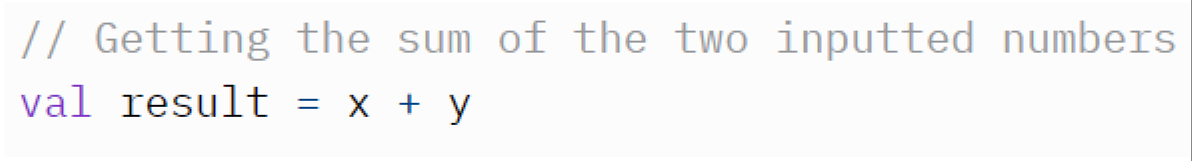


```
fun main(args: Array<String>) {
  val x = 1;
  val y = 10;

  if (x == 1) {
   if (y == 10) {
     print("Yay you picked the correct values!")
   }
 }
}
```

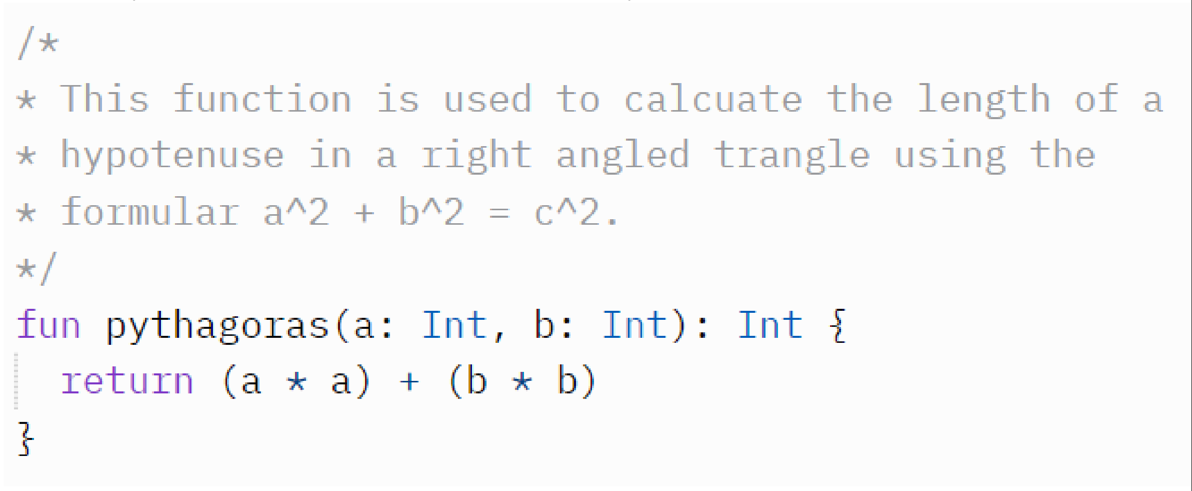
Comments

Comments are lines of human readable description that is used to describe the purpose of algorithms within your code. These comments are not read by the compiler or interpreter and are not compiled as part of your program.
There are 2 types of comments: a single line and blocks. A single line comment is usually used for shorter and less complex operations, such as basic math.



```
// Getting the sum of the two inputted numbers
val result = x + y
```

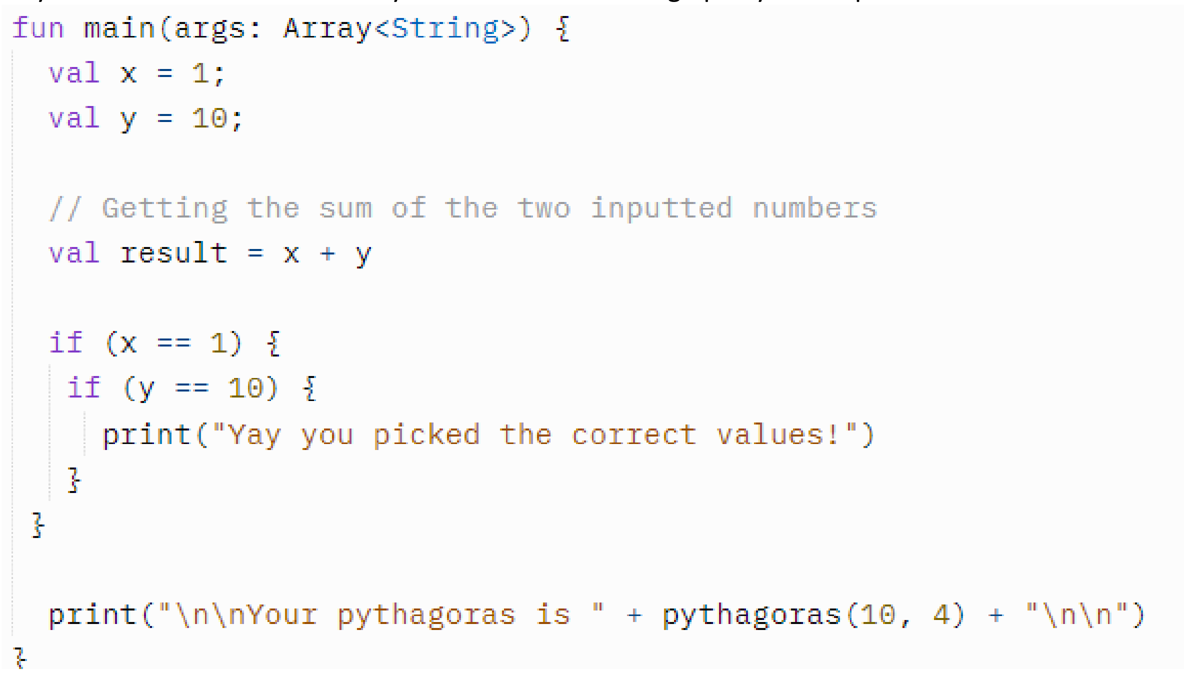
A blocked comment is a comment that spans over multiple lines of code. This is usually because the operation being described is more complicated. These comments are usually surrounded by a different operator, in our case "/* text here */" compared to "//".



```
/*
* This function is used to calcuate the length of a
* hypotenuse in a right angled trangle using the
* formular a^2 + b^2 = c^2.
*/
fun pythagoras(a: Int, b: Int): Int {
    return (a * a) + (b * b)
}
```

In most languages, comments are identified with "//". Anything after this is ignored by the compiler. When comment blocks are needed, "/*" and "*/" are commonly used in languages. This tells the compiler to ignore everything between those operators.

White space
Whitespace, which in simple terms are blank lines, are often used to help make code more readable as it allows programmers to space out their code. This is often done to separate functions, variables and different tasks within an algorithm. Whitespace is ignored by the compiler so you can have as much of it as you want without taking up any extra space.



```kotlin
fun main(args: Array<String>) {
    val x = 1;
    val y = 10;

    // Getting the sum of the two inputted numbers
    val result = x + y

    if (x == 1) {
      if (y == 10) {
        print("Yay you picked the correct values!")
      }
    }

    print("\n\nYour pythagoras is " + pythagoras(10, 4) + "\n\n")
}
```

Variable Names

When writing your code, you should be considered the names of the variables you are using to ensure that their purpose and function is clear and concise. When someone reads your code, they should be able to immediately understand the purpose of a variable. For example, whilst

```
a = b * c;
```

is valid code, it doesn’t tell us what the variables should be storing. Whereas

```
weekly_pay = hours_worked * hourly_pay_rate;
```

Is much easier to understand as it tells us what each variable is storing and what the type of data the outcome of the multiplication will be.

Each programming language has its own guides and recommendations for naming conventions. For example, C# suggests you should start private variables with an underscore, whilst Java suggests you shouldn’t start any variables with them. A lot of languages follow the camelCase naming convention, which is used to write phrases with any spaces. This is done by using a capital letter to indicate the start of the next word. For example, “eBay” and “iPhone” both follow this.