

Software Design and Development

# Assignment 1

Unit 6

George Hotten

## Data Types

Data Type	Description, example and memory
String	Collection of characters, for example "Hello world". This will take 10 bytes + the string length.
Integer	A whole number up to 2,147,483,647. For example, 420. This would take 4 bytes of memory.
Floating point	Decimal number with 1 decimal point, for example 6.9. This would take 2 bytes of memory.
Byte	Small integer between 0 and 255. For example, 69. This would take 1 byte of memory.
Date	Contains a date and sometimes a time. For example, "31/01/2022 13:51:52". This will take 8 bytes of memory.
Boolean	Has 2 values: true or false, 1 or 0, etc. Example is "true". This will take 2 bytes of memory
Single	Float but different name – 2 bytes of memory. This would take 4 bytes of memory.
Double	Decimal number with multiple decimal points. For example, 4.2069. this would take 8 bytes of memory.
Fixed Point	Decimal number with a fixed number of decimal points. For example, 4 with 3 fixed points is 4.000. 2 bytes + point length.

### Why are data types beneficial?

#### Memory Management

Data types are beneficial for memory management as the different data types take up different amounts in memory. For example, storing '25' as an integer would take up 4 bytes of memory compared to if it was stored as a byte, which would take up 1 byte of memory. This makes data types very beneficial and important for the optimization of your application and their performance. It also makes the application more accessible to less powerful systems as it isn't taking up any unneeded resources.

#### Data Validation

Data types are beneficial for data validation as the application can attempt to parse any inputted data to the required data type. If it fails, the data is invalid. This helps ensure the inputted data into an application matches the required type and range. This helps maintain the application's integrity as no foreign input can be used to break the program. It will also improve reliability as if the data is unable to be parsed and not handled correctly, the program will crash.

## Programming Constructs

### Selection

Selection is where certain code only executes if a condition is met. If the condition is not met, the program might terminate, or other code could be executed.

```
# Selection
name = input("What is your name? ")

if name == "George":
    print("Hello my lovely! 🥰")
else:
    print("Go away, impostor!!")
```

In this example, we request the user input of their name and store that in the variable "name". We then do an if statement to check who they are. If the variable "name" is equal to "George", hello will print. If not, go away will print.

The output of this program, if I entered "George" would be:

```
What is your name? George
Hello my lovely! 🥰
```

However, if I entered "Matt" the output would be:

```
What is your name? Matt
Go away, impostor!!
```

### Iteration

Iteration is where certain code is looped over until a condition is met, or for a certain number of times. For example, execute a block of code 10 times. Or execute a block of code whilst a condition is true.

```
# Iteration
number = 10

for n in range(1, number + 1):
    print("This has ran " + str(n) + " times!")
```

In this example, we assign the number 10 to the variable "number". Then using a for loop, we iterate over the print statement the number of times between 1 and the number in "number". This means the program will run 10 times.

The output of this program would be:

```
This has ran 1 times!  
This has ran 2 times!  
This has ran 3 times!  
This has ran 4 times!  
This has ran 5 times!  
This has ran 6 times!  
This has ran 7 times!  
This has ran 8 times!  
This has ran 9 times!  
This has ran 10 times!
```

### Sequence

Sequence is the set of instructions that must execute in a specific order, one before the other.

```
# Sequence  
num1 = int(input("Please enter the first number: "))  
num2 = int(input("Please enter the second number: "))  
  
print("The sum of the two numbers is " + str(num1 + num2))
```

In this example, we request 2 numbers from the user by storing them in the variables "num1" and "num2". We then print the sum of the 2 numbers by running `num1 + num2`. This must be executed in the order above or the program will not function correctly. For example, if we put the print statement first the program would error as `num1` and `num2` would not yet have been defined. For example:

```
# Incorrect Sequence  
print("The sum of the two numbers is " + str(number1 + number2))  
  
number1 = int(input("Please enter the first number: "))  
number2 = int(input("Please enter the second number: "))
```

Would cause an error at runtime as those 2 variables haven't been defined.

```
Traceback (most recent call last):  
  File "/Volumes/Saucey/Py/pytest/main.py", line 23, in <module>  
    print("The sum of the two numbers is " + str(number1 + number2))  
NameError: name 'number1' is not defined
```

However, if we keep everything in the correct order, the program will output:

```
Please enter the first number: 10  
Please enter the second number: 15  
The sum of the two numbers is 25
```

## Quality of Code

Whilst programming, you should keep in mind the quality of your code, this is to ensure it is efficient, useable, portable, and maintainable.

### Code Efficiency

Code efficiency is important as whilst creating your application as you should ensure that it is designed to run smoothly and to be as hardware efficient as possible by methods such as using proper garbage collection. You should ensure it will be compatible in different environments (such as in different operating systems) and ensure that it runs without faults (errors and warnings). This helps create a better user experience with the program as it will run smoother and will allow the program to be accessed by more people as it will be able to run on their system.

### Code Usability

Code usability is important as it helps other programs understand your code and allows them to start writing new code for it faster. You should ensure that your code is clear and concise, and every function and variable are used necessarily. Try to make your code as self-explanatory as possible so any future developers reading it can understand what it does and its purpose.

### Code Portability

Code portability is important as having programs that can run on multiple platforms will allow for a larger userbase and allow them to use your program wherever they want. Before starting your project, you should think about what environments you want your program to be able to run in. For example, code written in Object C cannot be executed in Windows, and code written on the .NET Framework cannot be executed on macOS, etc. This can help increase the productivity of the programmers and removes a lot of restrictions in their workflow.

### Code Maintainability

Code maintainability is important for the programmers who will maintain your code and program after you have stopped working on it. If your code is badly written, they will not be able to easily continue your work. Whilst writing your code, you should ensure it is easy for others to understand and later edit. If a function was hard right, include comments to explain how it works. When picking names for variables, functions, or namespaces, ensure they are self-explanatory. Make sure you do lots of testing to ensure there are no bugs which could throw off any future maintainers.

## Example

```
8 #include <iostream>
9
10 // Function asking for input of a user's favourite number
11 int getNumber() {
12     // Create the variable
13     int input;
14
15     // Output to the console asking for the input
16     std::cout << "Please your favourite number: ";
17
18     // Read and check the input
19     std::cin >> input;
20     if (std::cin.fail()) { // Will be true if the inputted data isn't an integer
21         std::cout << "That's not a number!" << std::endl;
22         return NULL;
23     }
24
25     // Return the inputted number
26     return input;
27 }
28
29 int main(int argc, const char * argv[]) {
30
31     // Allocate memory to the variable "num", with the value being requested from the getNumber function
32     int *num = new int(getNumber());
33
34     // Return error if null (code 1)
35     if (num == NULL)
36         return 1;
37
38     // Tell the user their inputted data
39     std::cout << "You inputted " << *num << "!" << std::endl;
40
41     // Free the allocated memory
42     delete(num);
43     |
44     return 0;
45 }
```

Here is an example of code that would be considered efficient, usable, portable and maintainable. It is efficient as it handles proper memory management, usable as the code is straight to the point using the simplest way of doing the task, portable as the C++ can be exported to run in all environments, and maintainable as it has comments, concise function/variable names and everything has been properly tested.

## Application and Limitations of Programming Paradigms

When choosing a programming language, you should consider what paradigm it comes under and whether that is suitable for your use case.

### Procedural

Procedural programming languages are languages that use an explicit sequence of instructions to complete a task. A few examples of this type of language are Visual Basic, Pascal, C and Fortran. Procedural is often used for basic applications, often found in the command line.

#### Advantages of Procedural

- Easier to define algorithms
- More beginner friendly
- Lower memory utilizations

#### Disadvantages of Procedural

- Harder to debug
- Longer and more complex code

### Event Driven

Event driven programming is where the flow of the program is driven by user actions, such as mouse clicks and key presses. This is usually found in languages such as JavaScript, TypeScript, ActionScript and C# (for coding in a game engine such as Unity). Event driven programming is often used for games, websites, user interfaces and consoles.

#### Advantages of Event Driven

- Programs are more interactive
- The hardware can interact with the software and vice-versa

#### Disadvantages of Event Driven

- Hard to master
- Flow of the program is hard to visualise and understand

### Object Oriented


Object Oriented programming is languages that use data objects which can store data in fields and code in functions and procedures. This is found in languages such as Java, Kotlin, C# and C++. Object Oriented programming can be used to create almost any application, from games to data stores to operating systems.

#### Advantages of Object Oriented

- Clear program structure
- Easy to maintain
- Easy to modify
- Easy to understand

#### Disadvantages of Object Oriented

- Steep learning curve
- Objects can sometimes interact in unexpected ways



# Why Choose a Particular Programming Language?



# Suitability in terms of features and tools

---

- When choosing a programming language to create an application, it is important that you consider the toolset and features of the language to ensure it is fit for its purpose. For example, for web apps you would choose JavaScript (and one of its many libraries) as it is the most supported. Then for game making, you would use C++ or C# as they can interact with libraries such as DirectX.
- Depending on the use-case, the scalability should also be considered to ensure it can handle higher loads. This would only matter for applications that will have hundreds or thousands of concurrent users.
- To keep things running smoothly, the efficiency and performance of the language should be considered. For example if you need high efficient code in terms of memory, you could use C++ as you can handle the garbage collection, unlike languages such as C# or Java.
- When launching your code into production, you should have a look at the language's ease of deployment – for example you should ensure your program can run in the environment you want it to. Running a C# program on Linux may not work depending on what version of .NET you are using.
- Finally, the productivity of the language should be considered – how long does it take to write code? How easy is it to debug? Etc.

# Availability of Trained Staff

---

- As learning a new language can take over 2 years, you should consider the availability of your staff and what languages they can code in. Working in a language you already know is also more productive meaning it will speed up development time and reduce the number of errors as they know what they are doing.
- You should also ensure that staff able to program in the chosen language will be available in the future. If not, you should know where to hire some to ensure the program can be maintained. It should be able to be maintained for at least 5 years into the future.
- Companies should ensure they maintain and audit for their staff's programming skills to ensure it meets the needs of the program and business they are running.

# Expandability

---

- If your application is going to be used on a large scale, you should ensure your program can scale up to demand to handle many concurrent users.
- You should ensure this is possible by seeing how scalable the chosen programming language is and consider the resources needed to handle the high number of users.
- Whilst the programming language must be able to handle the loads, the servers running the application must be capable too. You must ensure the server has the right amount of networking capacity and compute resources.
- Where possible, the capability of your application to scale should be tested prior to the application going into production. This can be done by simulating the expected load.
- As mentioned previously, the programming language used should be considered as some applications are more able to scale than others. For example. PHP supports 'clustering' where multiple servers work together to serve the users using the application.

# Reliability

---

- You should ensure the language you are going to be using is stable and is as up-to-date as possible. This lowers the likelihood of running into bugs and errors. A reliable language would have been well tested and have clear debugging and error handling.
- On the topic of error handling, the programming language should be able to handle errors without the entire programming crashing and it should be able to recover from any errors that do occur.
- If the application is not thoroughly tested, this will reduce the reliability of the program as it is more likely to break or run into unexpected loads and not know how to handle them.