Evaluation

Evaluation

When evaluating a piece of software we consider several areas:

- Fitness for purpose
- Efficiency
- Robustness
- Readability

Fitness for Purpose

Fitness for purpose answers the question "does your software meet the requirements identified during the Analysis stage of the software development lifecycle.

If you have a requirement to display information to the user, and your program doesn't do that, your program is not fit for purpose.

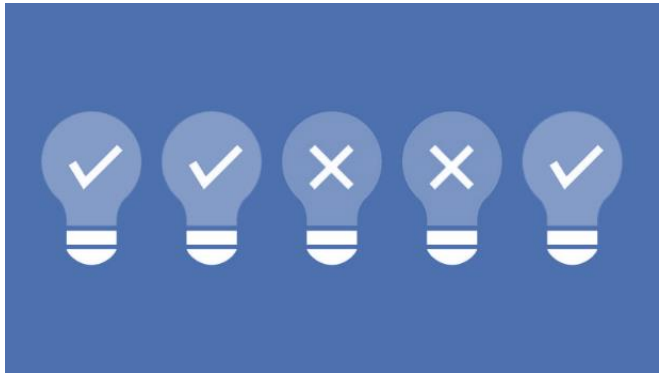
Efficiency

Efficiency is evaluated in two areas:

- How much processing power is required to run the code?
- How much time does it take the developer to write the code?

There are three constructs that we've looked at that make code more efficient.

- If statements
- Arrays
- Fixed loops



Efficiency: If Statements

Comparing these programs, we can see that Program 2 is more efficient.

Program 1 will **always** evaluate all three if conditions, even if the first one is evaluated to true.

Program 2 will not evaluate further conditions if the first condition is evaluated to true.

This reduces the number of instructions that need to be processed by the ALU.

```
#Program 1

if age >= 65:
   ticketCategory = "concession"
if age >= 18 and age < 65:
   ticketCategory = "adult"
if age <18:
   ticketCategory = "child"
```

```
#Program 2

if age >= 65:
   ticketCategory = "concession"
elif: age >= 18 and age < 65:
   ticketCategory = "adult"
else:
   ticketCategory = "child"
```

Efficiency: Arrays

Comparing these two programs, we can see that Program 2 is more efficient.

Program 1 requires many more lines of code to be written, which means more processing required during translation. It also requires more RAM to store the name and value of each piece of data.

```
#Program 1

#declaring variables
city1 = ""
city2 = ""
city3 = ""
city4 = ""
city5 = ""
city 6 = ""
city7 = ""
city8 = ""
city9 = ""
city10 = ""
```

```
#Program 2

#declaring variables
cities = [""] *10
```

Efficiency: Fixed Loops

Comparing these two programs, we can see that Program 2 is more efficient.

Program 1 requires many more lines of code to be written, which means more processing required during translation.

```
#Program 1
cities[0]=input("Enter city 1")
cities[1]=input("Enter city 2")
cities[2]=input("Enter city 3")
cities[3]=input("Enter city 4")
cities[4]=input("Enter city 5")
cities[5]=input("Enter city 6")
cities[6]=input("Enter city 7")
cities[7]=input("Enter city 8")
cities[8]=input("Enter city 9")
cities[9]=input("Enter city 10")
```

```
#Program 2
for counter in range(0,10):
    cities[counter]= input("Enter city " + str(counter))
```

Note: conditional loops should not be discussed in terms of efficiency as they do not replace an inefficient construct.

Evaluation: Robustness

Robustness refers to how well your program is able to cope with unexpected input. Unexpected data can be provided by files or directly from users.

Programs use input validation to ensure that users provide acceptable data, making them **robust**.

You can ensure that your program is robust by using exceptional test data during the Testing phase of development.

Evaluation: Readability

It’s important that your code is readable to help you and others understand what you intend your code to do.

Most of the time, code bases are written by dozens of people so it’s important that your code is readable so it’s easy for *any* member of the team to make changes and fix bugs.

Evaluation: Readability

Readable code should:

- Use meaningful variable names
- Use internal commentary to help explain complex parts of the code
- Have white space between different sections of code to separate different pieces of logic
- Use indentation to help to identify loops and selection statements