Notes: Programming languages

Programming languages similarly to human languages have a Grammer and syntax. While a specific programming language does not 'evolve' to learn new Grammer, subsequent languages that follow in its footsteps will build upon its successful features.

Programming languages allow computers to execute different tasks. Software features the term 'soft' because it is flexible and can be changed, unlike hardware which is ridged and unchangeable. In order for the tasks to be detailed in a way that computers can understand they must be written in a programming language that computers can execute.

At low-level computers still execute very simple operations such as bit instructions. Modern languages allow a higher level of thinking but still compile down into the same simple instructions. Programming languages are abstractions to make it easier to create programs.

Compiler vs Interpreter

Complied languages are compiled once before execution can occur. Compiling involves three steps. Tokenizing the source code. This involves analyzing and creating a token representation of the system. The 'tokens' are then optimized in step two. Finally, the tokens are generated into the target code language. Compiled code must be re-complied entirely any time the code changes.

Interpreted languages do not compile code before execution, they instead tokenize the code piece by piece as the code runs. The interpreter only produces the tokens, a separate engine is responsible for executing the tokenized code for each platform.

Compiled code executes fastest, but interpreted code are often more platform independent.

Some languages - such as C++ - have static typing. Static typing requires the types of variables to be declared before compilation/interpretation. This can allow additional checks to be make on the types leading to more secure software.

Programming paradigms

There are four umbrella categories for programming paradigms:

Imperative - code is executed in a sequential order.

Functional - The structure of the code is composed of functions - purely functional code only has deterministic functions

Object Oriented - The code is structured as a set of objections with behavior and properties.

Logical - Answers are determined based on databases of knowledge.

These are just a broad grouping and there are many more sub paradigms and combinations.

History

In 1948 the small-scale experimental machine executed its first program.

In 1940 the first assembly language was written.

In the 1950s the first 'modern' programming languages such as Fortran where written.

Pythons' language features

Abstraction - Abstraction allows code to hide how its implemented

Casting - Casting allows datatypes to be converted into other datatypes for example a float into a int.

Encapsulation - Encapsulation allows developers to hide data and implementation details.

Garbage Collection - Garbage collection automatically deletes unused resources, freeing up memory.

Grammer - The python grammar is human readable

Inheritance - Child classes can inherit functionality from parent classes

Mutability - Objects cannot be changed. If for example a number is changed from a 2 to a 1. the variable is the same but the object changes.

Polymorphism - The use of inheritance allows for polymorphism where the functionality called in independent of the object passed in.

Recursion - Python allows recursion. Functions can call themselves.

Coding practices - Bad

Code smells - certain 'anti-patterns' can be examples of potential problems.

God objects - This is a problem where one object has to many responsibilities.

Refused Bequest - If a class breaks the inheritance contract it causes problems.

Long method - functions that are two long and should be split into multiple functions.

Parameter creep - to many parameters can indicate a function does to many different things.

Cyclomatic Complexity - to many branches and loops. Hard to understand and test.

Poor naming - makes code hard to understand

Design Patterns

Design pattern can be used to solve frequently encountered problems in programming. Using design patterns creates a consistent set of approaches to solving common problems. Design patterns encourage a modular development approach.

The three categories of design patterns: "Creational design patterns provide solutions to support the creation of objects. Structural design patterns support the organism"