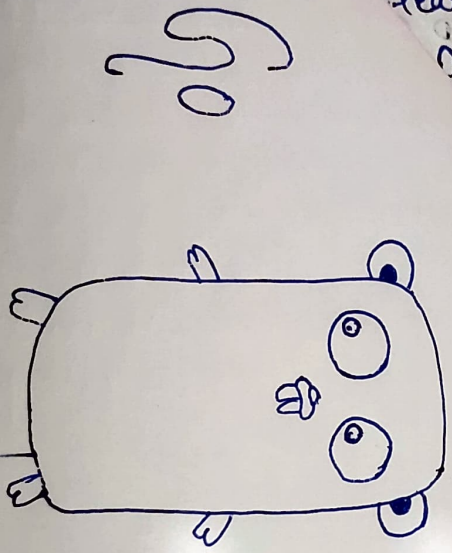
Go is an open source programming language developed
Google in 2007. It was designed to be simple, efficient
to read and, with strong support for concurrency
working.

Go :-

an syntax: Go's syntax is straightforward,
learn and write.

Go compiles directly to machine code,
by fast and efficient.

t: Go provides built-in support
go-routines and channels,
that utilize multiple-core



y management
with a rich standard
is to cryptography.
ps catch errors at compile time.
easily compile code for
tectures.

golang:

execution model: Go uses the 'go'
compile source code into executable binary. This
machine code specific to the target architectures.
require runtime environment.
ries are statically linked by default, meaning
the necessary libraries and dependencies are included
in the final executable. This results in a standalone
binary that can be easily distributed and run on any
compatible system.

Go lang

Go is an open source programming language developed by Google in 2007. It was designed to be simple, efficient and easy to read and with strong support for concurrency and networking.

Key features of Go :-

- Simple and clean syntax : Go's syntax is straightforward, making it easy to learn and write.
- Compiled language : Go compiles directly to machine code, which makes it extremely fast and efficient.
- Concurrency control/support : Go provides built-in support for concurrent programming through go-routines and channels, making it easy to write programs that utilize multiple-core processors effectively.
- Garbage collected : Automatic memory management
- Strong standard library : Go comes with a rich standard library that ranges from web servers to cryptography.
- Static Typing : Type safety helps catch errors at compile time.
- Cross platform compilation : easily compile code for different OS and architectures.

Internal working of golang.

Compilation and execution model : Go uses 'the go' command to compile source code into executable binary. This binary contains machine code specific to the target architectures. It does not require runtime environment.

Go binaries are statically linked by default, meaning all the necessary libraries and dependencies are included in the final executable. This results in a standalone binary that can be easily distributed and run on any compatible system.

Go makes cross compilation straightforward. We can compile a Go program for different O.S and architectures by setting environment variables like GOOS (target OS) and GOARCH (target architecture).

Memory management in Go

Go uses garbage collection to manage memory. Go's GC automatically handles the allocation and deallocation of memory, reducing the risk of memory leaks.
Go lang uses escape analysis to decide where to store the data in memory - stack or the heap. Stack is fast and small and variables whose context is limited to a function are allocated stack memory. While heap is the larger area of memory used for allocating variables that needs to live longer. It requires more work from the GC to manage.
GC is optimized for low latency applications. It runs concurrently with the application's code, meaning it performs garbage collection without pausing the entire program.

Goroutines and concurrency model.

Goroutines are like tiny, super-efficient threads that let you run multiple things at once in your Go program.

- Lightweight → These are very small and don't take up much memory. We can run thousands or even millions of them running at the same time without slowing down the computer too much. This is different from traditional threads in other languages, which are heavier and can slow down your system if you've too many.
- Automatic management : → They don't require us to manage them closely. Go runtime takes care of starting, stopping and switching b/w them.

- Each goroutine starts with a small bit of memory and grows or shrinks as needed. Traditional threads have fixed size stack.
- Scheduler tries to use computer's resources efficiently by running multiple go routines on the available CPU cores.
- Channels are used by go routines to communicate with each other. They don't require locks or other complex synchronization mechanisms.
- Go's runtime is responsible for managing low level tasks like goroutine scheduling, memory allocation and garbage collection. Runtime is part of every Go program and is statically linked into the final library.
- Go compiler is highly optimized to produce binaries quickly, even for large codebases. This makes Go an excellent choice for rapid development and iteration.

Installation

- Download go lang from golang.org. Install
- Create a file with extension → go.
- Install a go module in your directory using → go mod init hello.
- Install VS code extensions for go.
- To run the file → go run file_name.go

- Functionalities are written in functions. Similar or same kind of fns are grouped together to form a package. These packages are exported as modules so that the fns can be used by everyone. Just import the package/module and call its fn. Ex → Quote-Package.

How to import a package and use it.

1. Import it in your main file
2. On terminal run → go mod tidy. It will try to find the package you mentioned in your import → update go.mod file and add go.sum file.
3. If it exists, no error will be thrown. Now, run the file.

Modules

Packages and modules in depth

A package is a way to organise and reuse code. It's a coll^n of related go source files that are grouped together under a single package name. Each go file has package declaration at start that specifies the package it belongs to. ex → utils package for utility fns.

Module is a collection of related packages. It is defined by a go.mod file, which contains the module path and dependencies it requires.

*** Go expects all files in a directory to belong to the same package. Go compiles all files in a directory together.