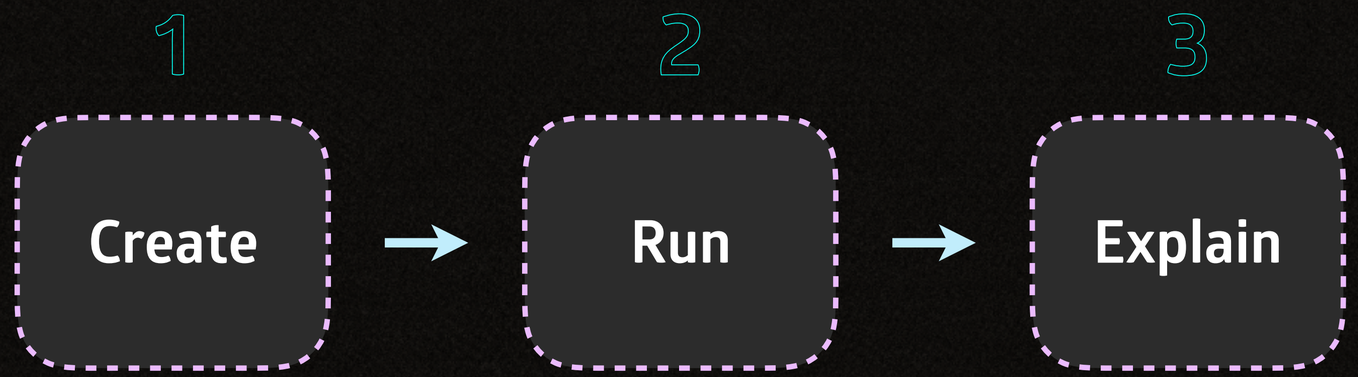
FIRST PROGRAM

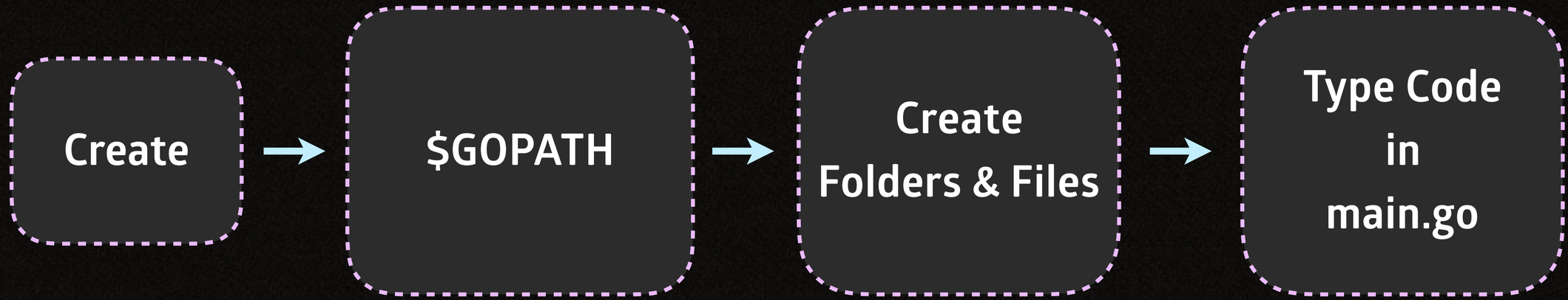
FIRST PROGRAM

You're going to learn how to create and run your first Go program



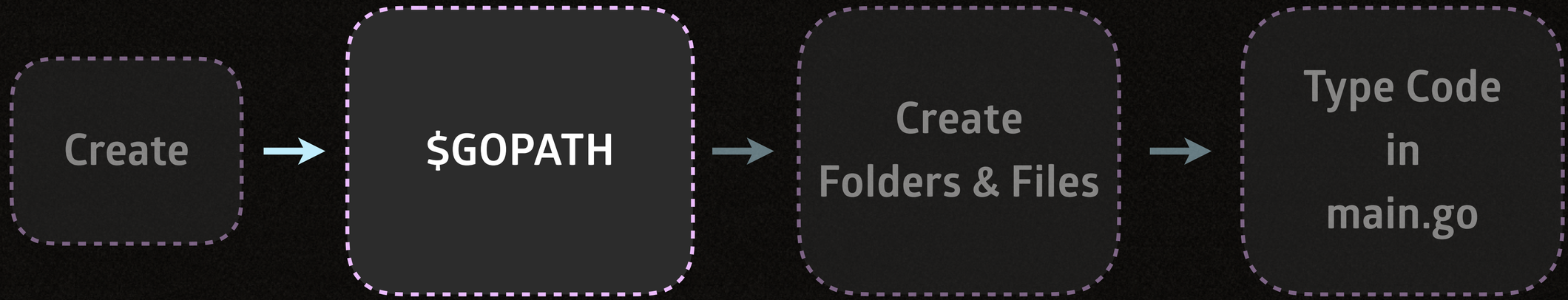
CREATE

In this first step, you're going to create your first Go program.



LEARN ABOUT $GOPATH

You're going to create folders for your program



$GOPATH

demonstration

$GOPATH

an environment variable

by default Go assumes that it's in your **Users folder**

do not set your *$GOPATH* variable

$GOPATH

workspace

Go **tools** find Go **source-code** files

by looking at $GOPATH

$GOPATH

physical folder

automatically **creates** a **physical folder** for *$GOPATH*

*under your **users** folder*
(its default location)

$GOPATH

src folder

contains source-code files

$GOPATH

src folder

Go source code file

an executable go program
needs its own directory

$GOPATH

src folder

Go source code file

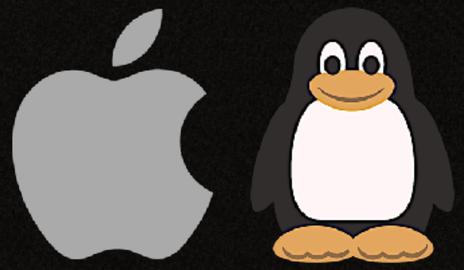
$GOPATH

- /src
  - /github.com
    - /youruser
      - /first
        - /main.go

GOPATH

Where is your $GOPATH? Let's find it!

$GOPATH



~/go

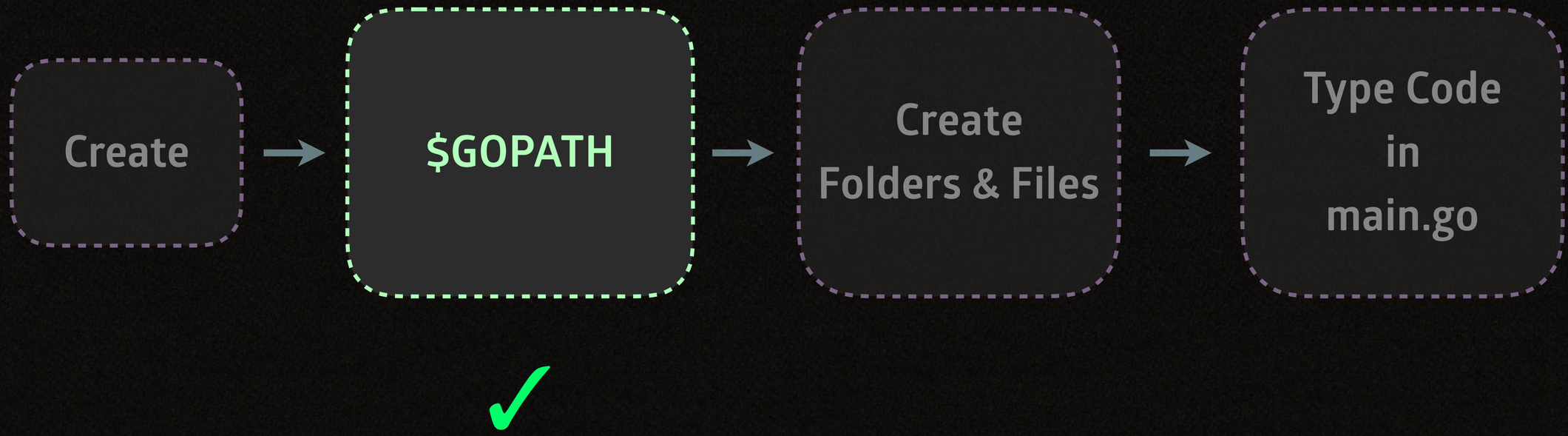
/Users/YourName/go

%userprofile%\go

c:\Users\YourName\go

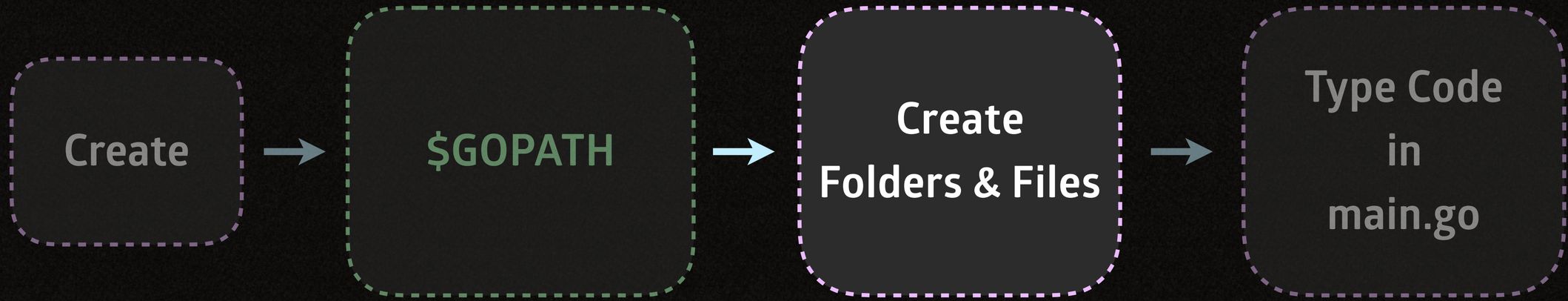
GOPATH

You've learned about $GOPATH and src folder and you've opened it in VS Code. Happiness.



CREATE FOLDERS & FILES

You're going to create folders and main.go



CREATE FOLDERS

Create these folders under your $GOPATH/src

1. learngo
2. first

If you've a **github** account:

$GOPATH/src/github.com/username/learngo/first

DEMO TIME!

CREATE MAIN.GO

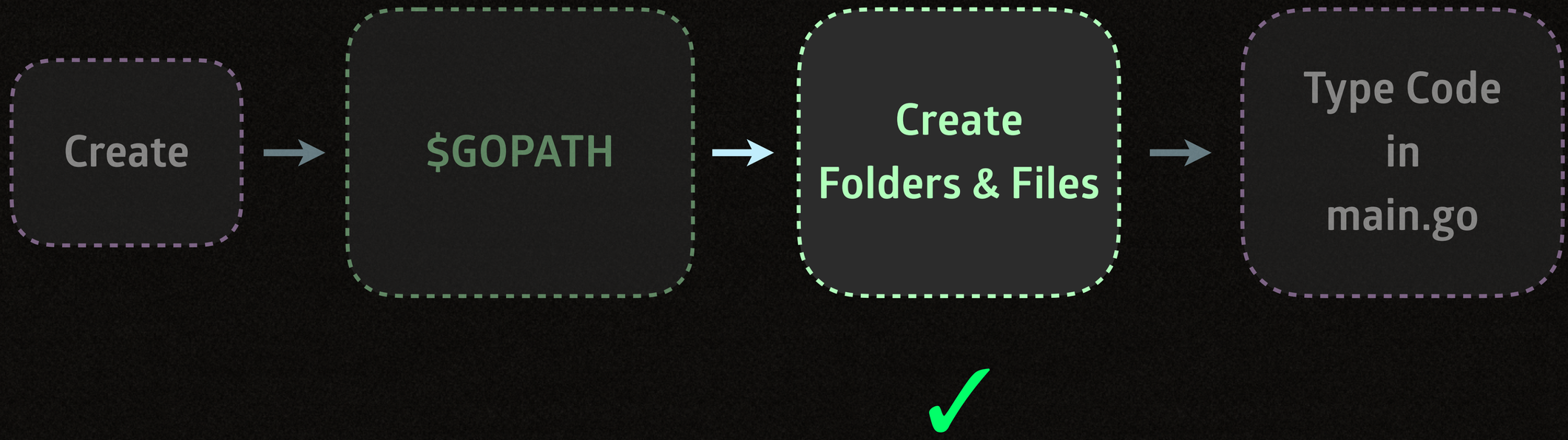
Create your first program file: "main.go"

under:

$GOPATH/src/github.com/username/learngo/first

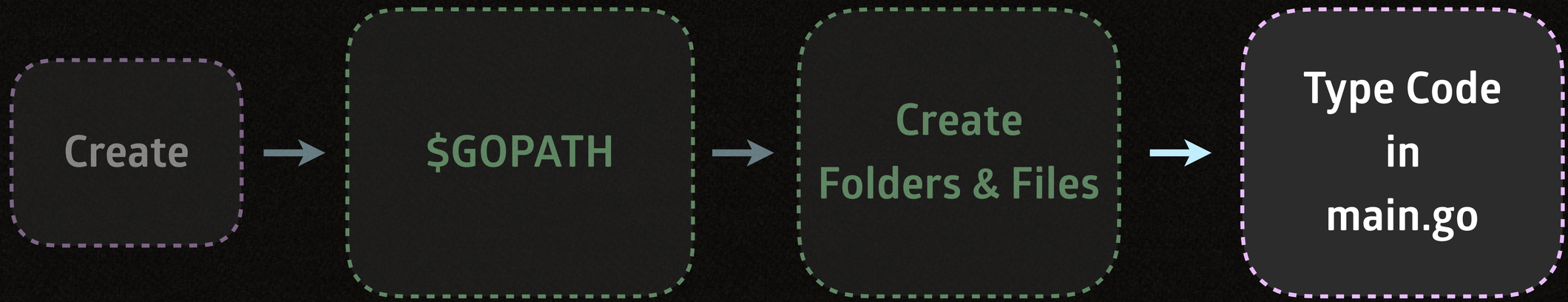
CREATE MAIN.GO

You've created your first Go file, congrats!



CODE

You're going to code your first Go program



CODE

Let's code your first Go program

main.go

```
package main

func main() {
    function's code
    will be in here
}
```

package clause
this should be the first code

***name** of the package*
which this file belongs to

function
reusable and executable block of code

func main
a special func that tells Go where to start executing

CODE

I've removed the annotations from the code for clarity

main.go

```go
package main

func main() {
}
```

CODE

Let's print something to the console!

```go
package main
import "fmt"

func main() {
  fmt.Println("Hello Gopher!")
}
```

HOW TO CODE WITH ME

I'll always be providing the full source-code for the examples

sometimes,
i won't show these
common statements
in here

```
package main
import "fmt"

func main() {
}
```

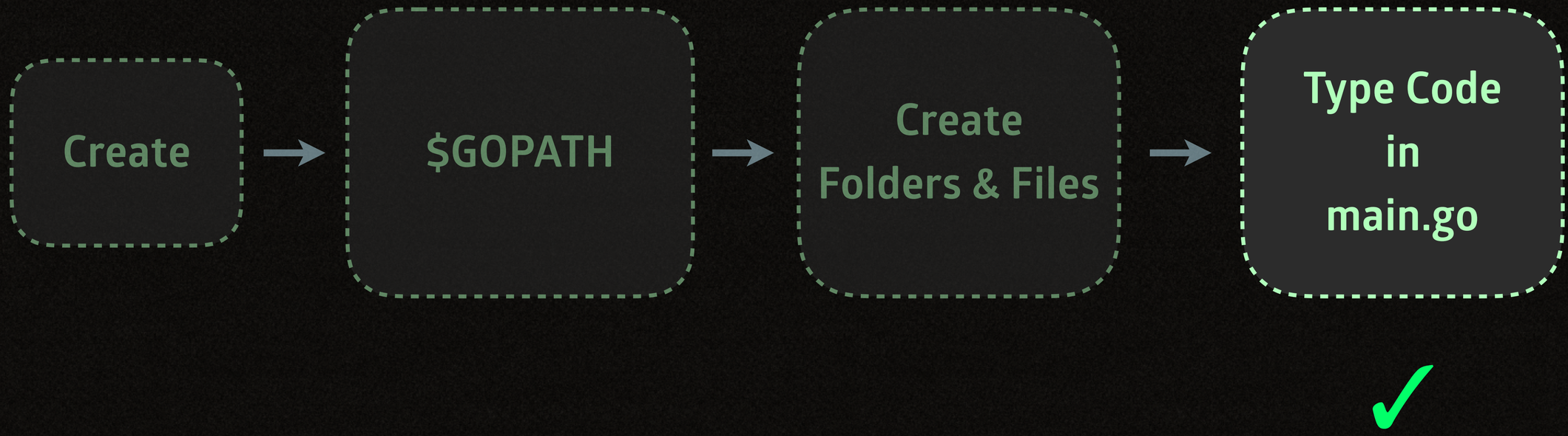
full source-code
will always be
available on
github
&
playground

github
https://github.com/inancgumus/learngo

playground
https://play.golang.org/

CREATE

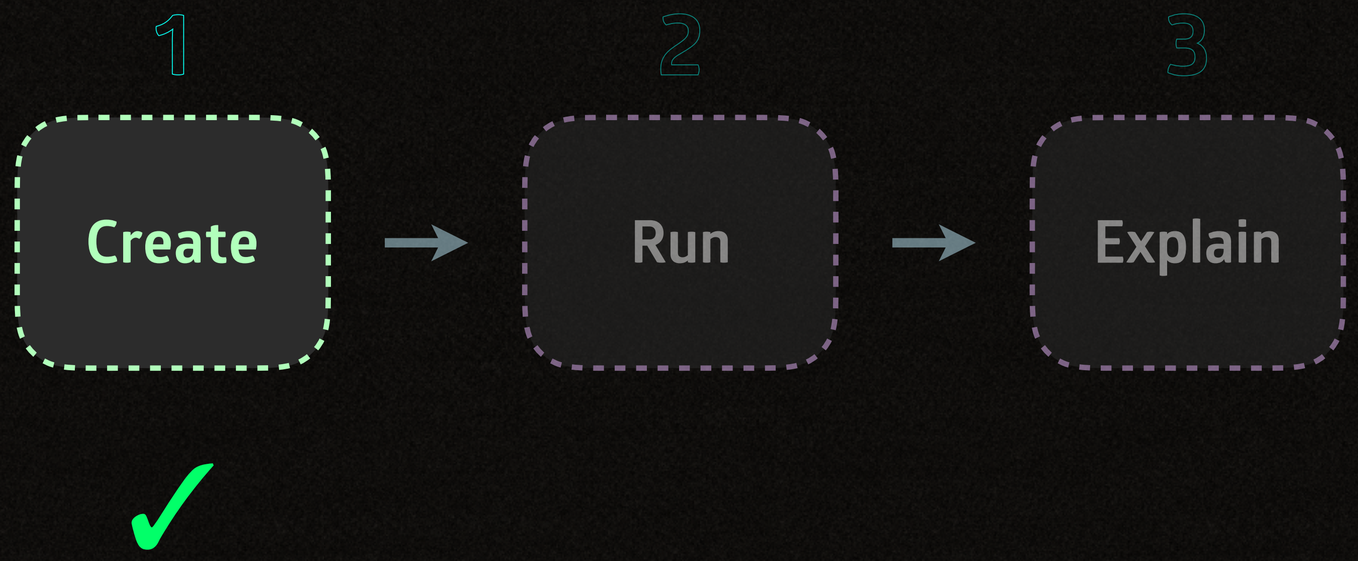
You've completed creating your first program, well done!



✓

FIRST PROGRAM

Congrats! You've learned how to create your first Go program.



RUN

You're going to run your first Go program.

GO BUILD

You're going to learn how to compile your Go program using `go build` tool.

go build

./first

first

DEMO TIME!

COMPILE-TIME

vs

RUNTIME

SOURCE CODE

```
package main
import "fmt"

func main() {
   fmt.Println("hello!")
}
```

$ go build

Compiling

COMPILED CODE

```
MOVQ GS:0x8a0, CX
CMPQ 0x10(CX), SP
JBE 0x108e7c7
SUBQ $0x48, SP
MOVQ BP, 0x40(SP)
...
```

Source-Code is like a seed
(compiled or not)

RUNTIME

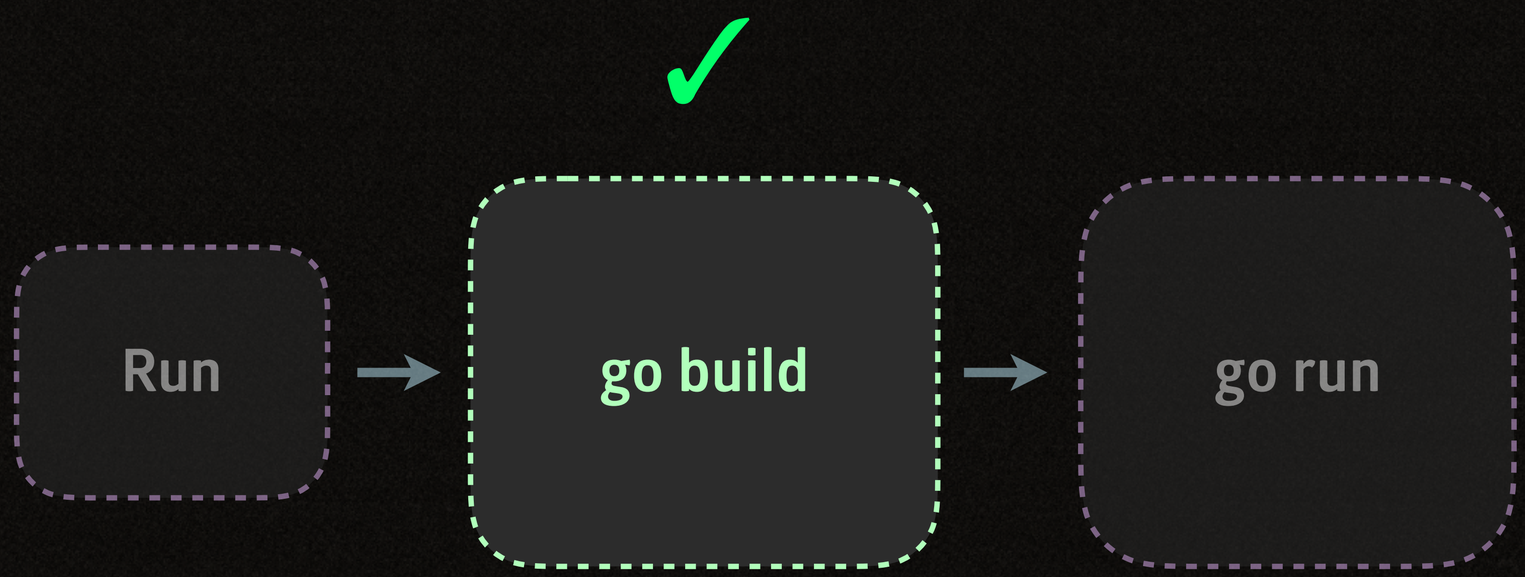
memory

cpu

```
$ ./first
hello!
```

GO BUILD

You've learned how to compile and run your first Go program using `go build` tool.



GO RUN

You're going to learn how to compile your Go program using `go run` tool.

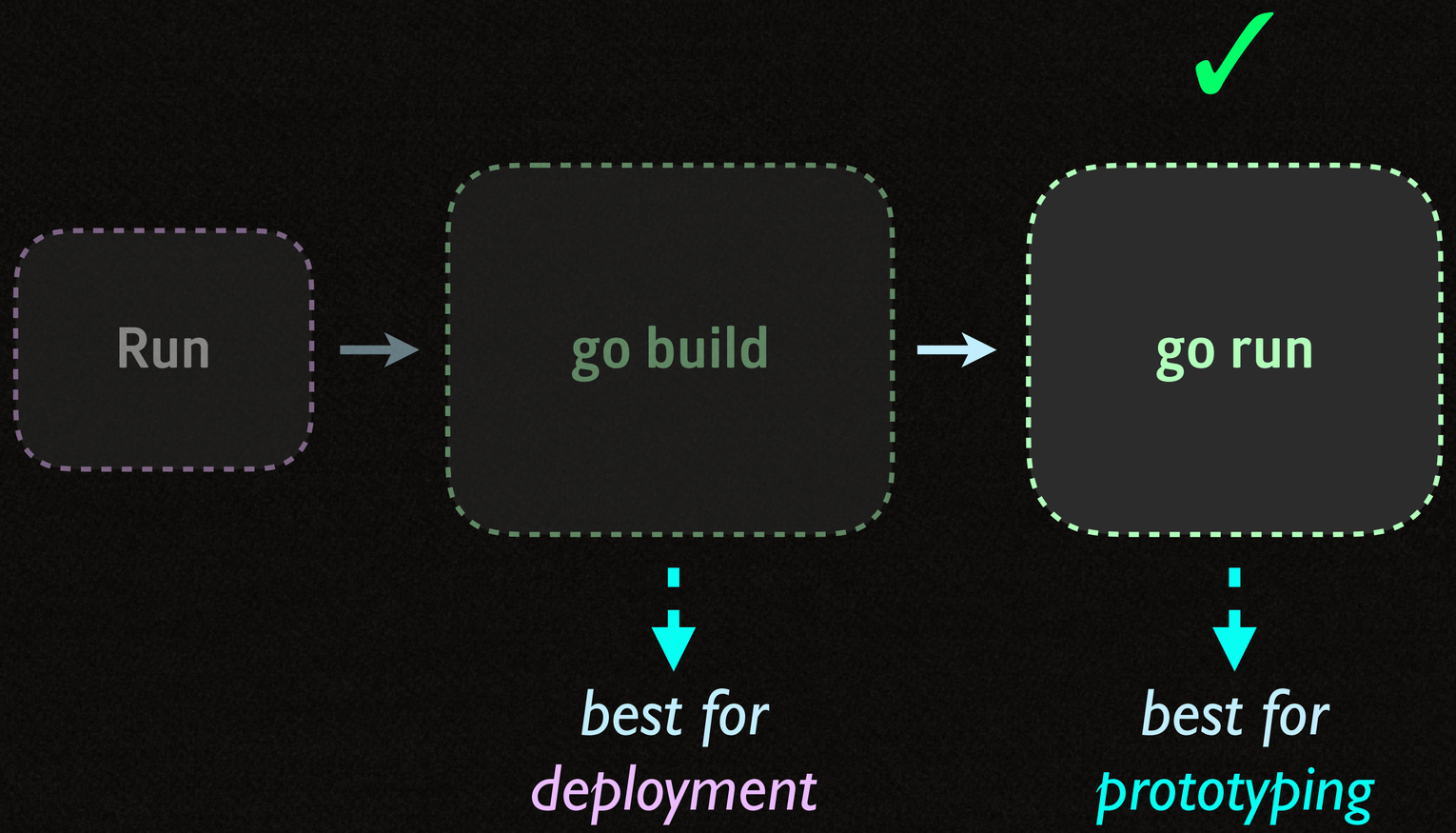

```
go run main.go
```

DEMO TIME!


$ go run main.go
Hello Gopher!
I love animations
And colors...
> Please embrace this

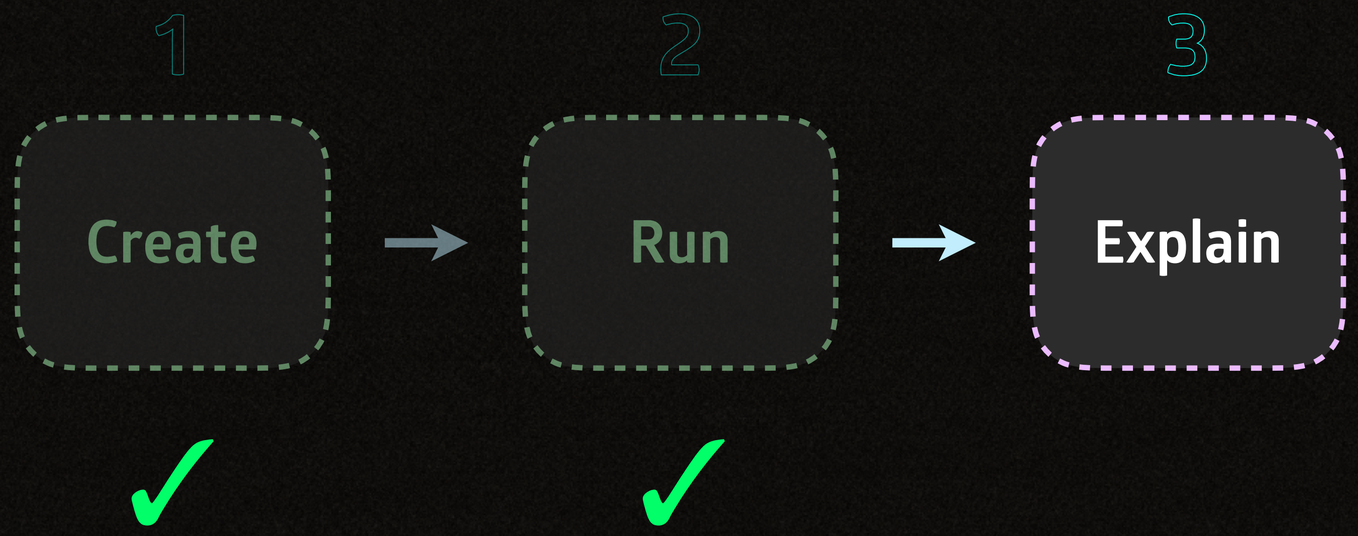
GO RUN

You're going to learn how to compile your Go program using `go run` tool.



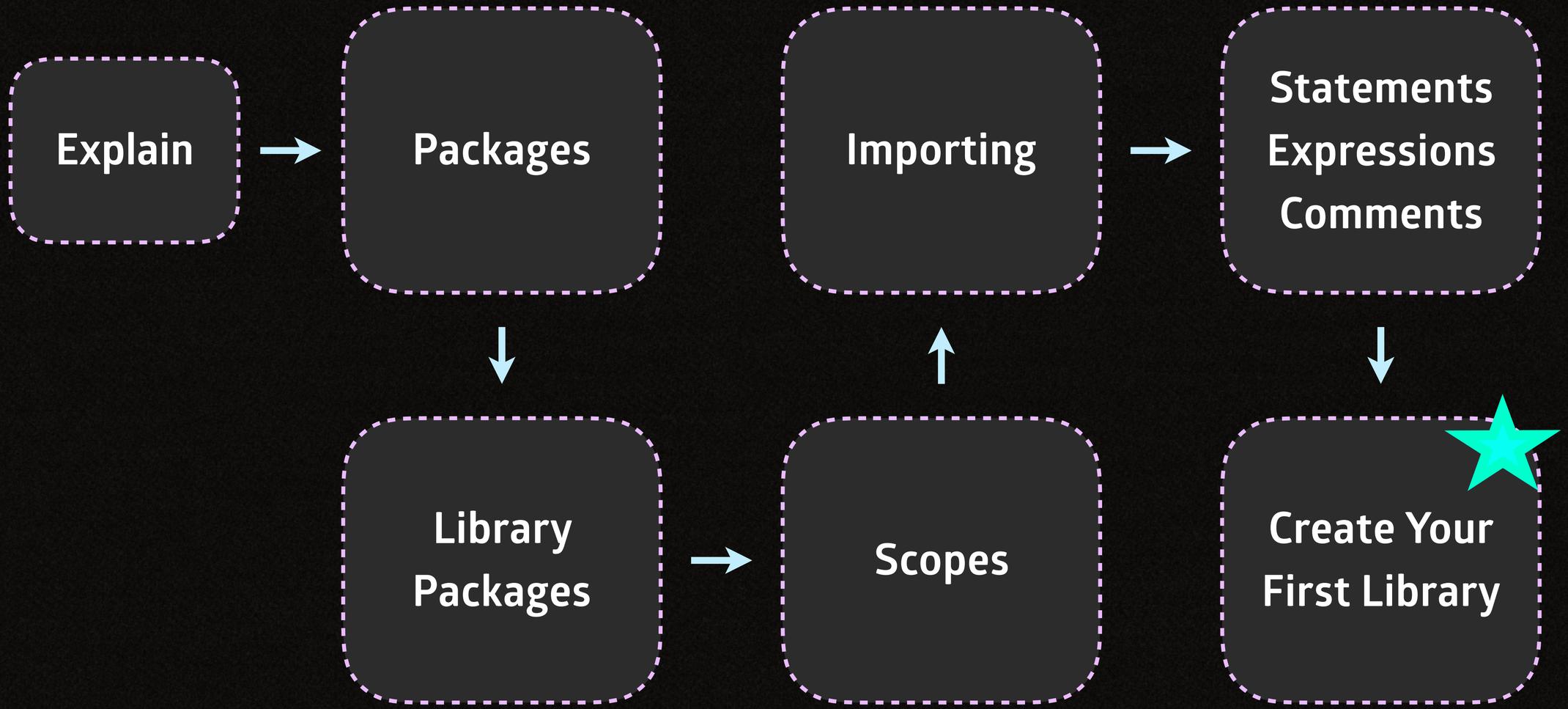
FIRST PROGRAM

Congrats! You've learned how to compile and run your first program! That's a great achievement!



EXPLAIN

There are a lot of new knowledge awaits you!



PACKAGE

all **package files** should be in **the same directory**

example

$GOPATH/src/github.com/username/learngo/first/explain/packages/what

PACKAGE

all **files** should **belong** to **the same package**

package main
package x
package y

PACKAGE CLAUSE

You can use package clause in a file to let Go know that which package that file belongs to.

package clause ◀- - -

this should be the first code

it can only appear once

```
package main
import "fmt"

func main() {
  fmt.Println("Hello!")
}
```

PACKAGES

"main.go" file belongs to the main package.

main.go

```go
package main
import "fmt"

func main() {
   fmt.Println("Hello!")
}
```

learngo/first/explain/packages/what

PACKAGES

"bye.go" file belongs to the main package.

bye.go

```go
package main
import "fmt"

func bye() {
  fmt.Println("Bye!")
}
```

learngo/first/explain/packages/what

PACKAGES

"hey.go" file belongs to the main package.

hey.go

```go
package main
import "fmt"

func hey() {
  fmt.Println("Hey!")
}
```

learngo/first/explain/packages/what

PACKAGES

All of the files belong to the package main. Each file has a package clause as "package main".

learngo/first/explain/packages/what

they all belong to
package main

PACKAGES

Each Go package has its own scope.
For example, declared funcs are only visible to the files belong to the main package.

all the files are inside the
same directory
learngo/first/explain/packages/what

all the files belong to the
same package

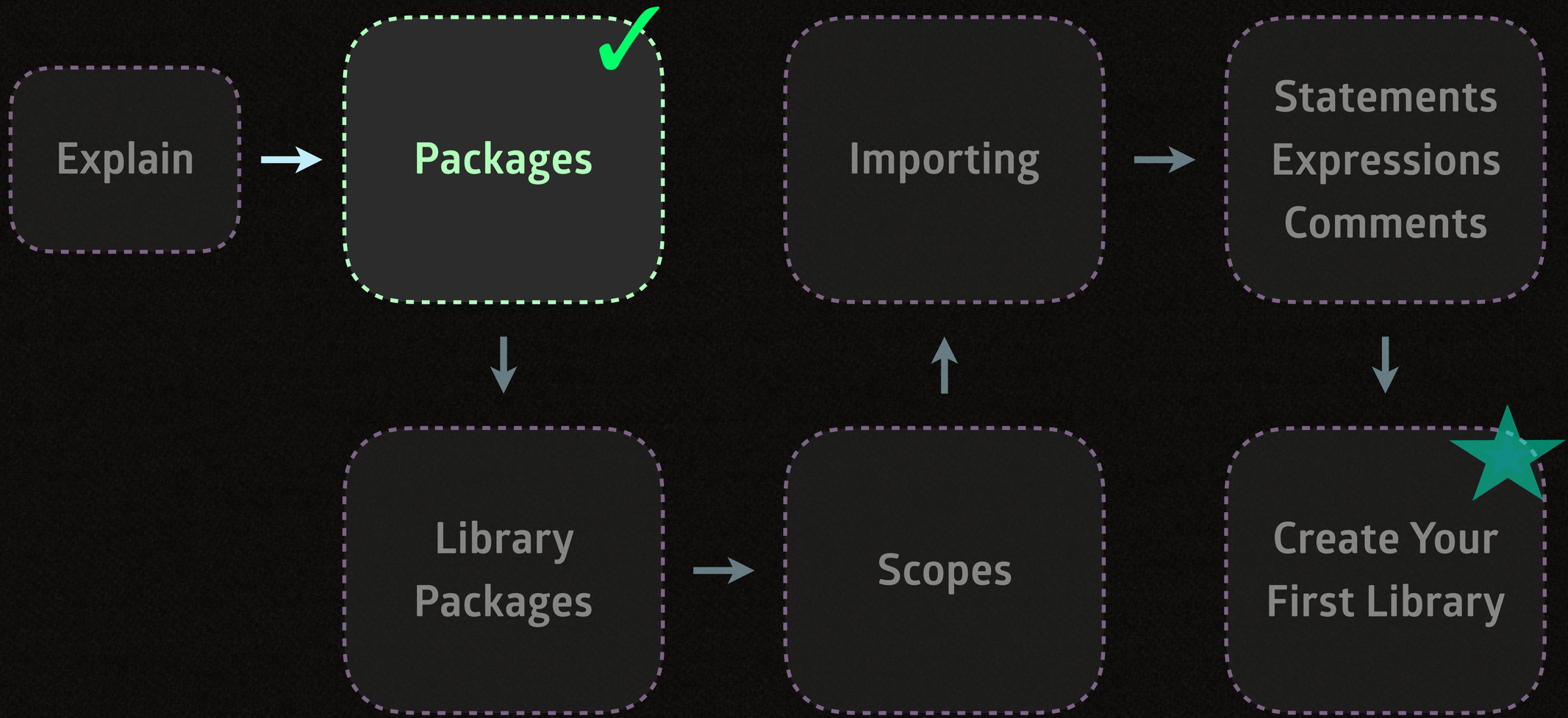
PACKAGE #2

demonstration

CODING TIME!

EXPLAIN

Congrats! You've completed the first step.



EXECUTABLE

vs

LIBRARY

PACKAGE KINDS

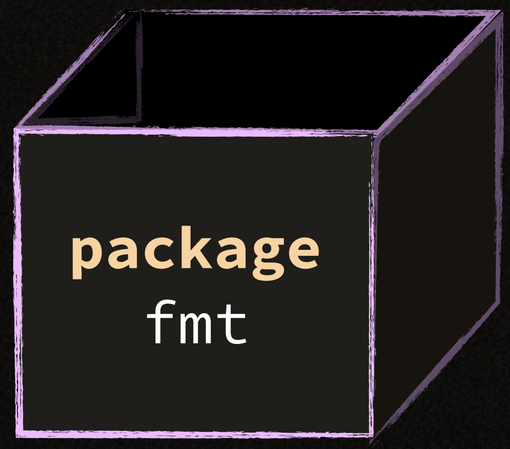
There are two kinds of packages in Go: Executable and Library.

Executable Package

+

func main( )

Library Package



EXECUTABLE PACKAGE

an **executable go program** should belong to **package main**

go build

go run

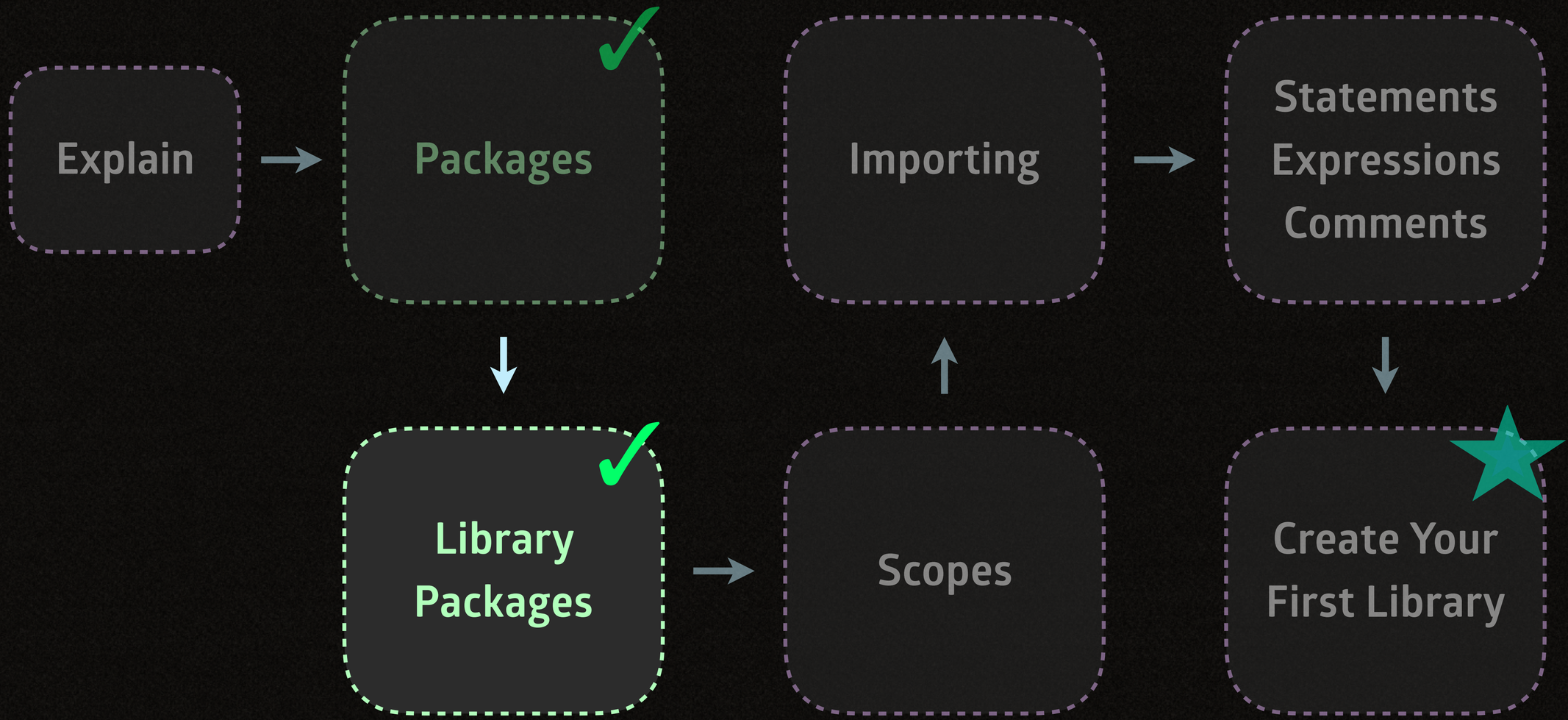
LIBRARY PACKAGE

created for reusability

LIBRARY	EXECUTABLE
created for **reusability**	created for **running**
non-executable	executable
importable	**non-importable**
can have **any name**	name should be **main**
no **func main**	**func main**

EXPLAIN

Congrats! You've completed the second step.



SCOPE

who can see what

there are **package**, **file**, **func** and **block** scopes

there are a few more...

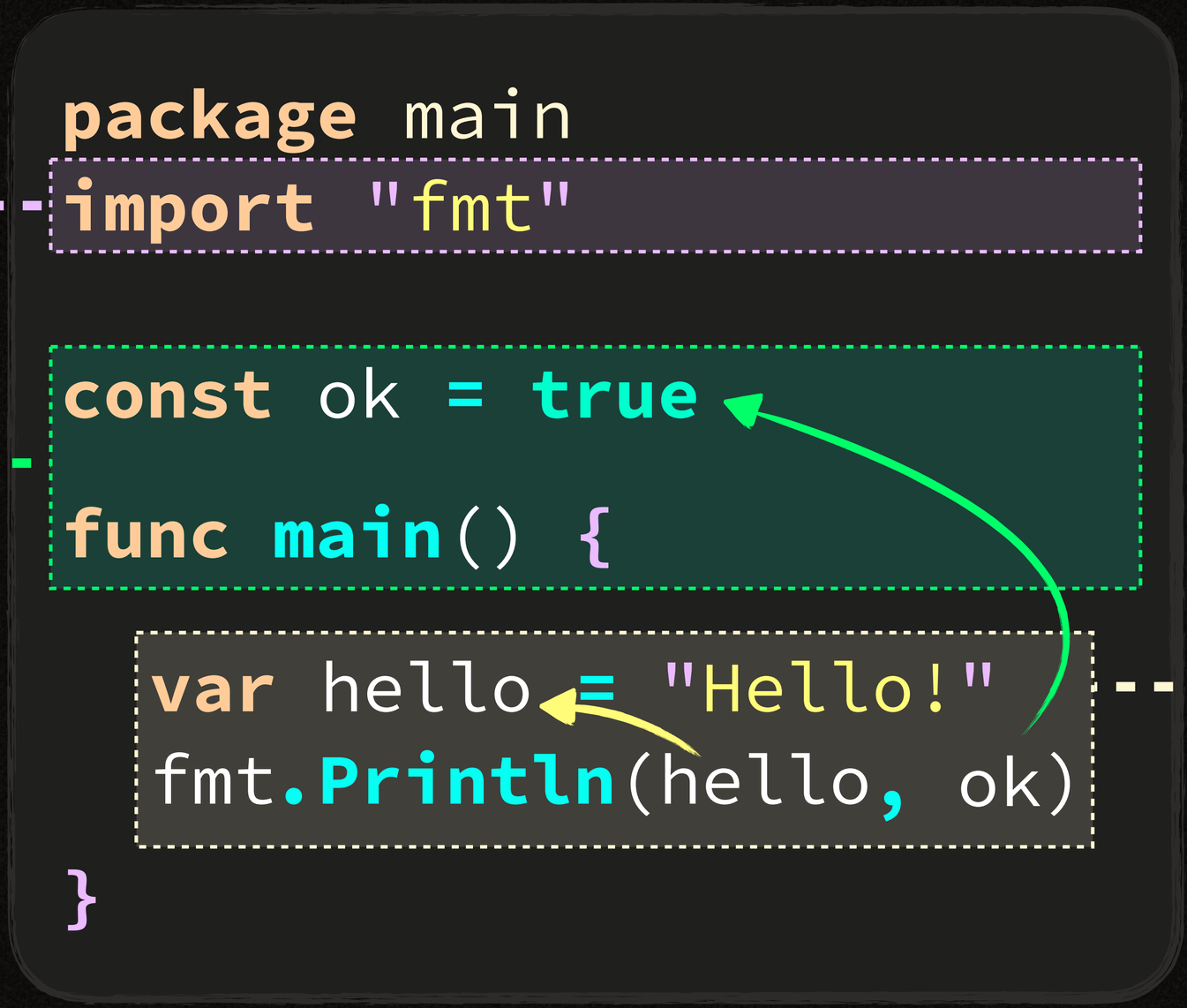
DECLARATIONS

declares a unique name bound to a scope

same name *cannot be declared again* **inside the same scope**

SCOPE

Every line of code can have different scope depending on their position in a Go file.



file scoped

only visible in this file

package scoped

visible to all the files belong to the package

other packages can't see them

block scoped declaration

only visible after its declaration until " } "

SCOPE

Every line of code can have different scope depending on their position in a Go file.

```
package main
import "fmt"

func nope() {
    const ok = true
    var hello = "Hello!"
    _ = hello
}

func main() {
    fmt.Println(hello, ok)
}
```

block scoped declaration

only visible in "nope" func

PACKAGE SCOPE

names are visible throughout the package

PACKAGE SCOPE

Each Go package has its own scope. For example, declared funcs are only visible to the files belong to the same package.

*"**main**()" is visible throughout the main package*

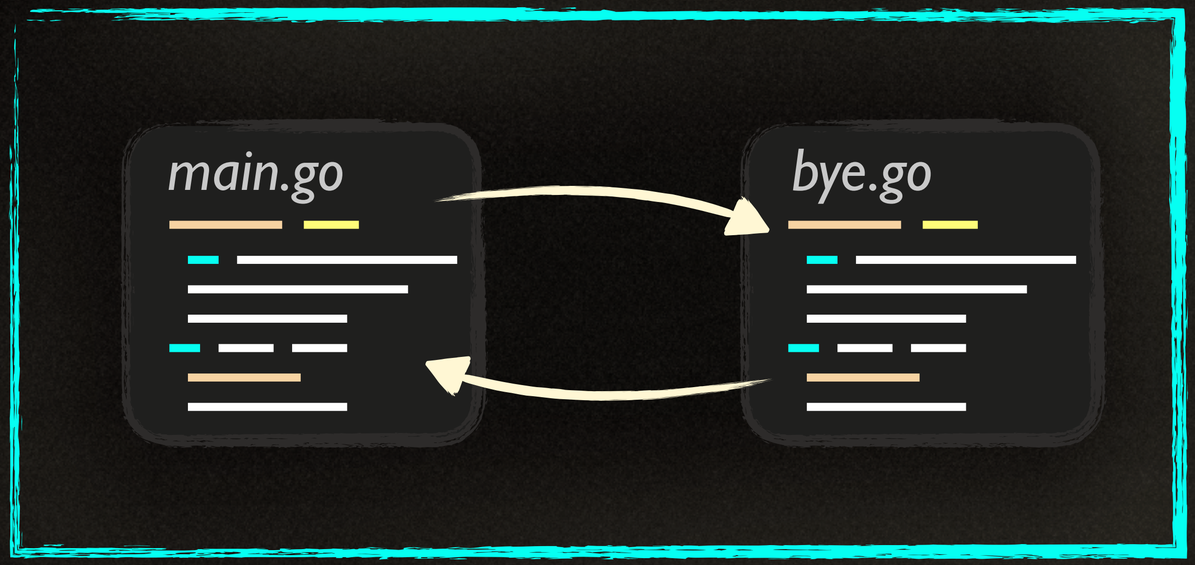
```go
package main
import "fmt"

func main() {
    fmt.Println("Hello!")
}
```

PACKAGE SCOPE

Declarations which are outside of functions are visible to the files belong to the same package.

main package



PACKAGE SCOPE

Each Go package has its own scope
For example, declared funcs are visible in the same package

main.go

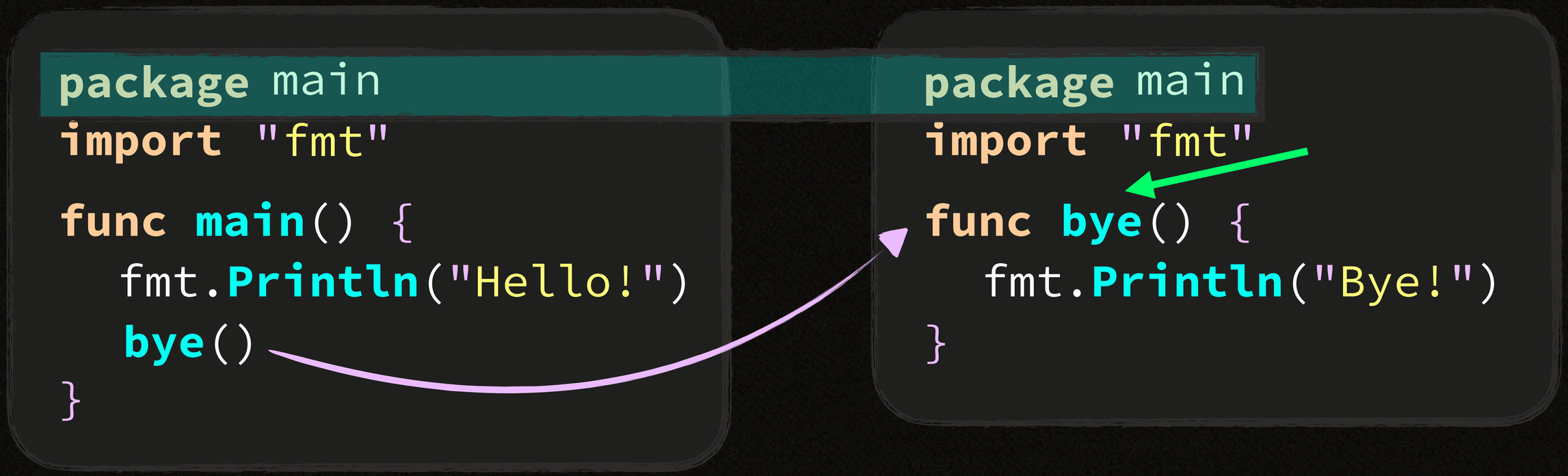
```go
package main
import "fmt"

func main() {
    fmt.Println("Hello!")
    bye()
}
```

bye.go

```go
package main
import "fmt"

func bye() {
    fmt.Println("Bye!")
}
```



CODING TIME!

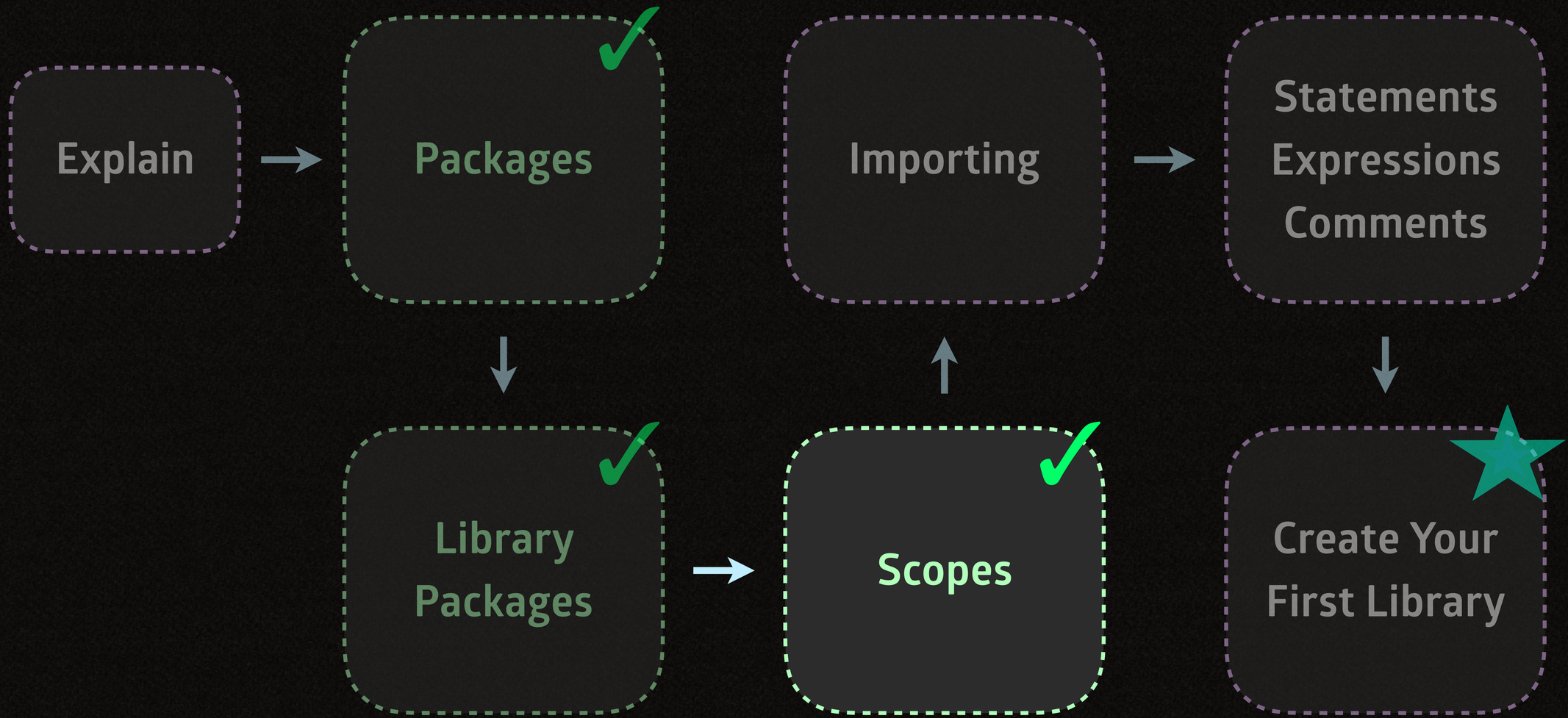
PACKAGE SCOPE #2

declaring the same names in the same scope

CODING TIME!

EXPLAIN

Congrats! You've completed the third step.



IMPORTING #1

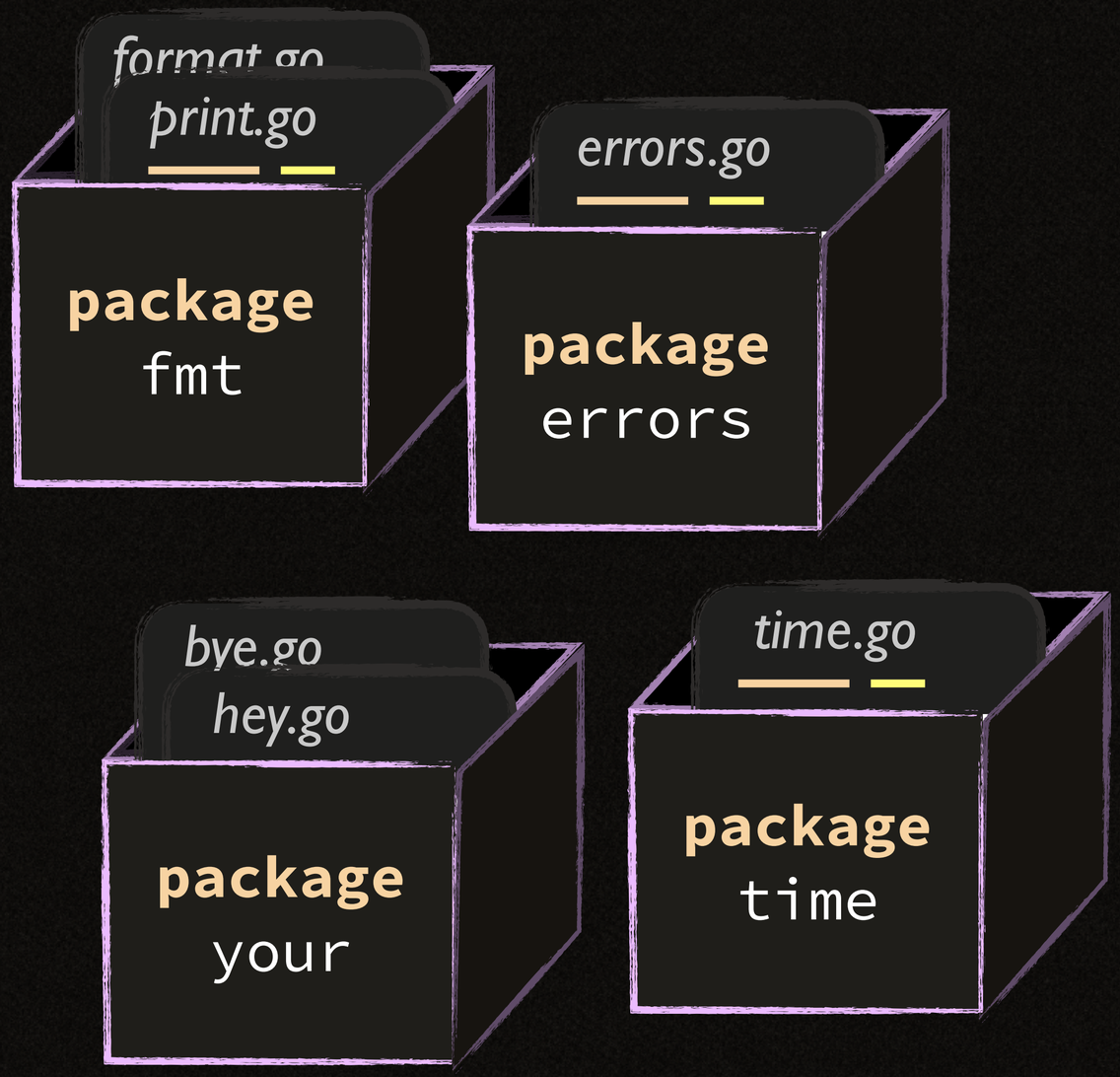
allows a **file** to use **functionalities** from **a library package**

IMPORTING

Importing is like as if you've declared what's inside the imported package's files in your own file.

myfile.go

```
import "fmt"
import "errors"
import "time"
```



FILE SCOPE

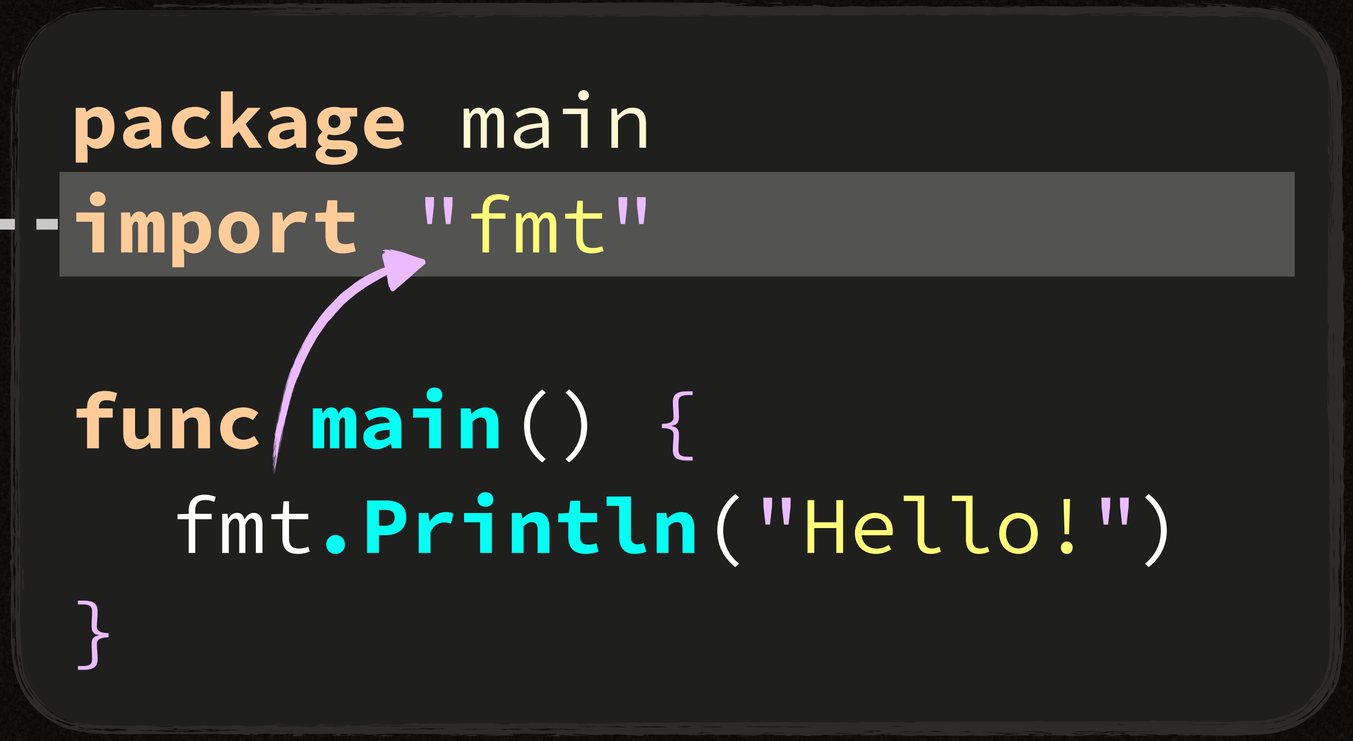
names are visible throughout the file

FILE SCOPE

Each Go file has its own scope
Imported packages are only visible to the importing file

main.go

"***fmt***" *is visible throughout the file*



FILE SCOPE

Each Go file has its own scope
Imported packages are only visible to the importing file

bye.go

```go
package main

func bye() {
    fmt.Println("Bye!")
}
```

bye.go can't use a package that it didn't import

FILE SCOPE

Each Go file has its own scope
Imported packages are only visible to the importing file

✓

```
package main
import "fmt"

func bye() {
    fmt.Println("Bye!")
}
```

FILE SCOPE

Each file has to import external packages on its own

main package

learngo/first/explain/importing

IMPORTING #2

you can **rename** the name of a declared package

RENAMING IMPORTS

Multiple import declarations should be unique (in the same Go file).

within the same scope, any new declarations require unique names

RENAMING IMPORTS

You can import packages with the same name into the same file by giving one of them another name

```
package main
import "fmt"
import f "fmt"

func main() {
    fmt.Println("Hello!")
    f.Println("There!")
}
```

now you can use fmt package using these names:

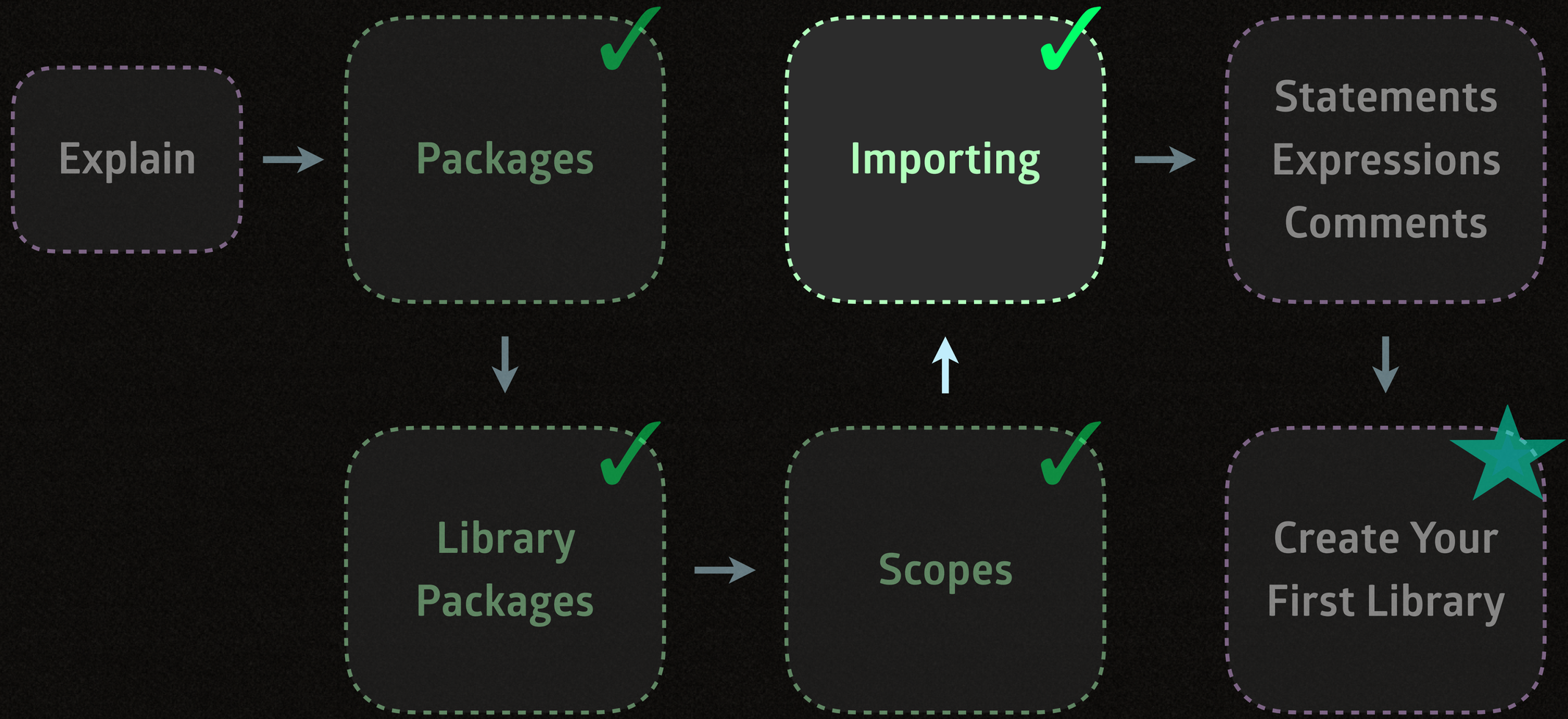
fmt

or

f

EXPLAIN

Congrats! You've completed the fourth step.



STATEMENTS

instructs Go to execute something

one statement at a time
(*unless you use semicolons* **;**)

STATEMENTS

Statements tell Go to execute something
and they can change the execution flow of code

```
package main
import "fmt"

func main() {
   fmt.Println("Hello!")
}
```

STATEMENTS

Almost all the code are statements here
They can stand on their own on a single line of code

```
package main
import "fmt"

func main() {
   fmt.Println("Hello!")
}
```

STATEMENTS

Statements control the execution flow

```go
package main
import "fmt"

func main() {
   fmt.Println("Hello!")
}
```

STATEMENTS

Statements control the execution flow. Look at how if statement changes the execution.

```go
package main
import "fmt"

func main() {
    fmt.Println("Hello!")

    if 5 > 1 {
        fmt.Println("bigger")
    }
}
```

STATEMENTS

Now, there are two statements: both are fmt.Println function calls.

```
package main
import "fmt"

func main() {
    fmt.Println("Hello")
    fmt.Println("World!")
}
```

STATEMENTS

Go adds a semicolon between statements, behind the scenes.

```
package main
import "fmt"

func main() {
   fmt.Println("Hello"); fmt.Println("World!")
}
```

There are still two statements here

EXPRESSIONS

computes one or multiple values

expressions should be used with or within a *statement*

some statements like func calls can also act like expressions

EXPRESSIONS

Expressions calculate (evaluate) and return one or more values (multiple).

```go
package main
import "fmt"

func main() {
   fmt.Println("Hello!")
}
```

EXPRESSIONS

There's only one expression here: "Hello!" string literal.

```
package main
import "fmt"

func main() {
    fmt.Println("Hello!")
}
```

EXPRESSIONS

Operators allow you to combine expressions.

```
package main
import "fmt"

func main() {
   fmt.Println("Hello!" + "!")
}
```

"Hello!!"

EXPRESSIONS

Functions can be used as expressions inside statements.

```
package main
import (
    "fmt"
    "runtime"
)

func main() {
    fmt.Println(runtime.NumCPU())
}
```

EXPRESSIONS

You can use operators with function call expressions.

```
package main
import (
    "fmt"
    "runtime"
)

func main() {
    fmt.Println(runtime.NumCPU() + 1)
}
```

EXPRESSIONS

You can use operators with function call expressions.

```
package main
import (
   "fmt"
   "runtime"
)

func main() {
   fmt.Println(runtime.NumCPU() + 1)
}
```

EXPRESSIONS #2

demonstration

CODING TIME!

COMMENTS

documentation

```
// single line comments

/*
  multi line comments
  more than single line
*/
```

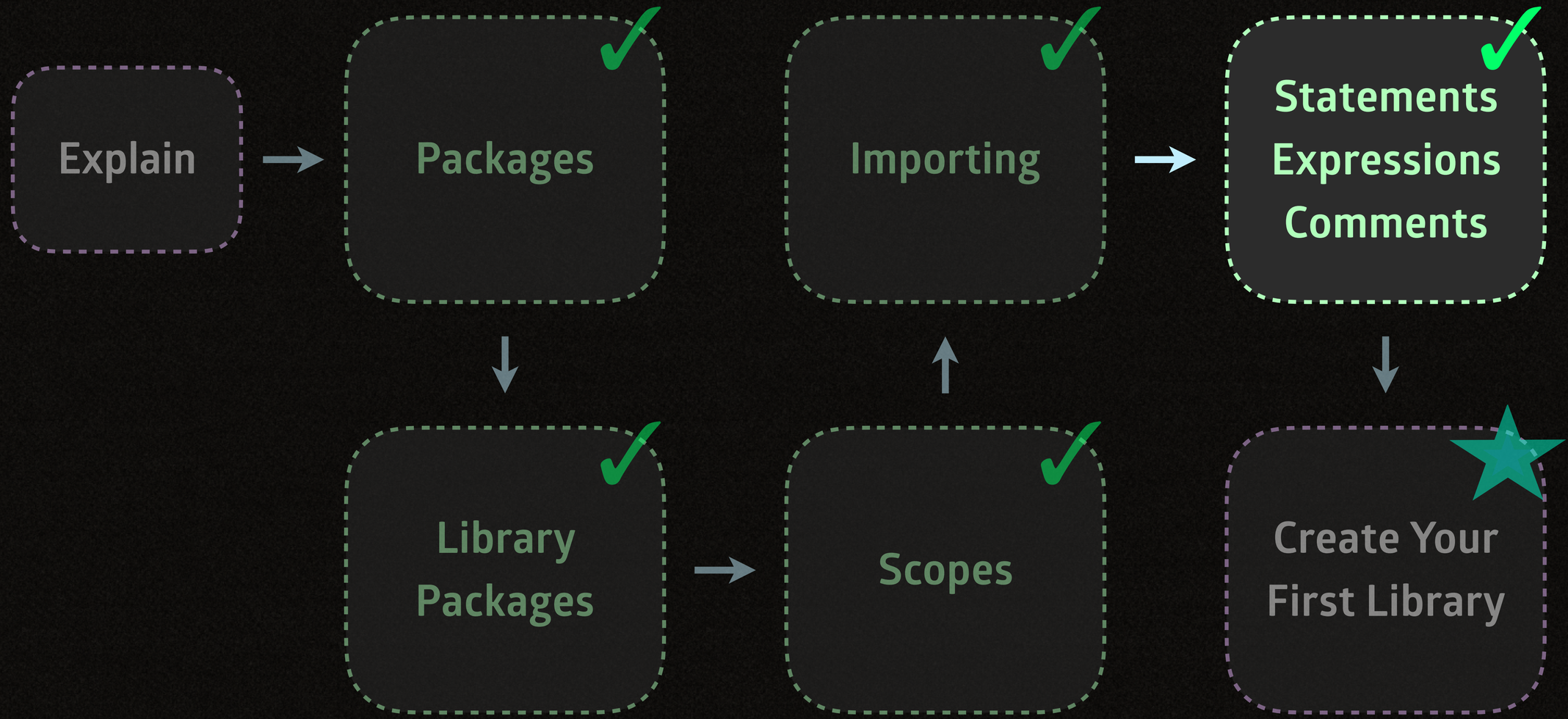
CODING TIME!

GO DOC

creates documentation automatically

EXPLAIN

Congrats! You've completed the fifth step.



CREATE YOUR OWN LIBRARY PACKAGE #1

CODING TIME!

CREATE YOUR OWN LIBRARY PACKAGE #2

EXPORTING

allows a **package** to make its **functionalities**
available to **other packages**

to export a name just make its first letter an uppercase letter

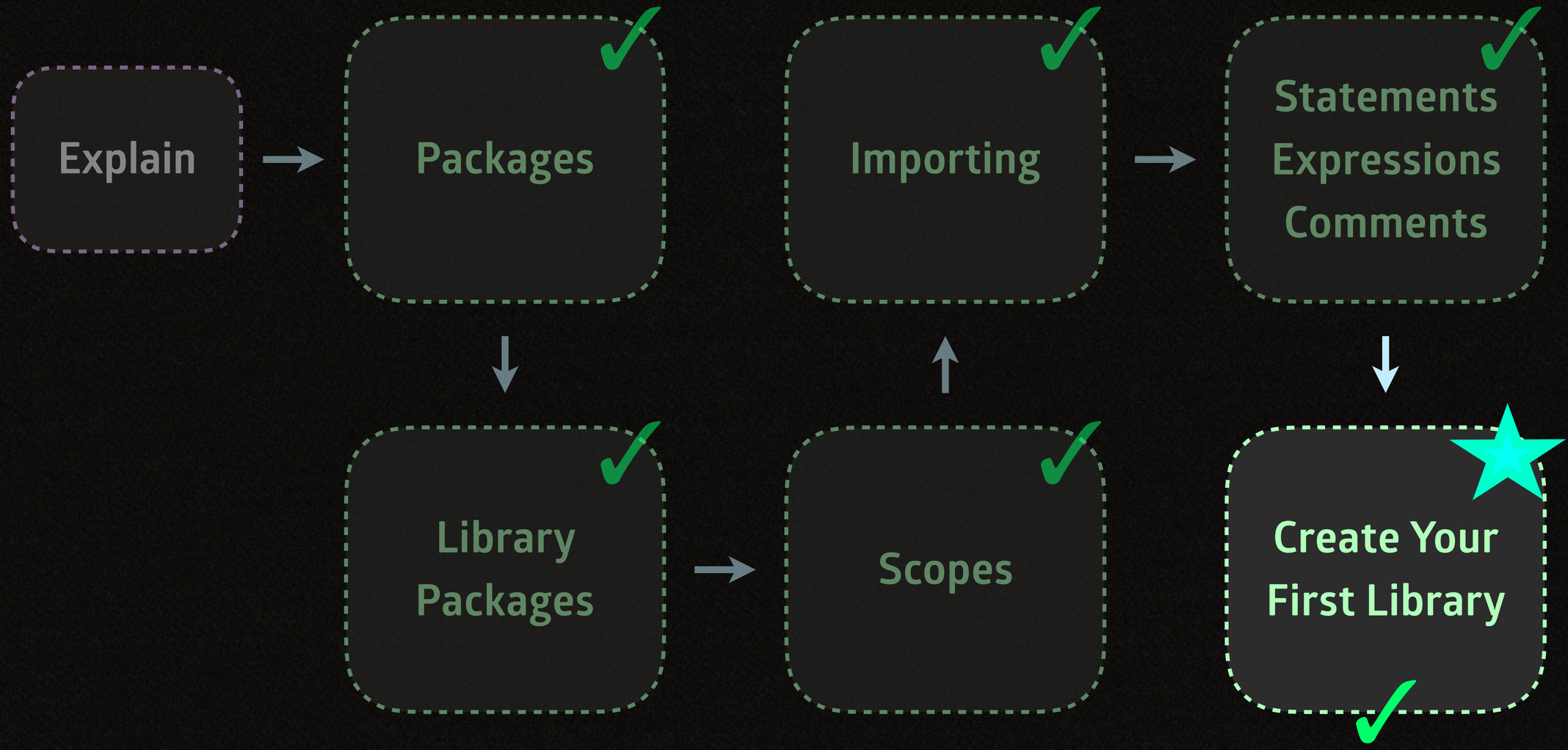
DEMO TIME!

CREATE YOUR OWN LIBRARY PACKAGE #3

CODING TIME!

EXPLAIN

Congrats! You've completed the fifth step.



DONE DONE DONE

Congrats! Now you're ready to be a Gopher! You're awesome!

1 Create ✓

2 Run ✓

3 Explain ✓