Go Track
Wk 1 Practical

NGEE ANN POLYTECHNIC

Program Structure

My First Go Program

Description	*In this lesson, you will learn the basic structure of a Go program.*
Learning Objectives	• How Go program is structured • How to run Go program • What are the key components of a Go program
Duration	30 minutes

Structure

A hello program is provided below to show the different components that make up the basic structure of a Go program.

Hello program code snippet:

```
//hello.go

package main

import "fmt"


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

In Visual Studio Code, in the Go workspace, create a new file under src, name it as hello.go, and type in the above Hello world program:

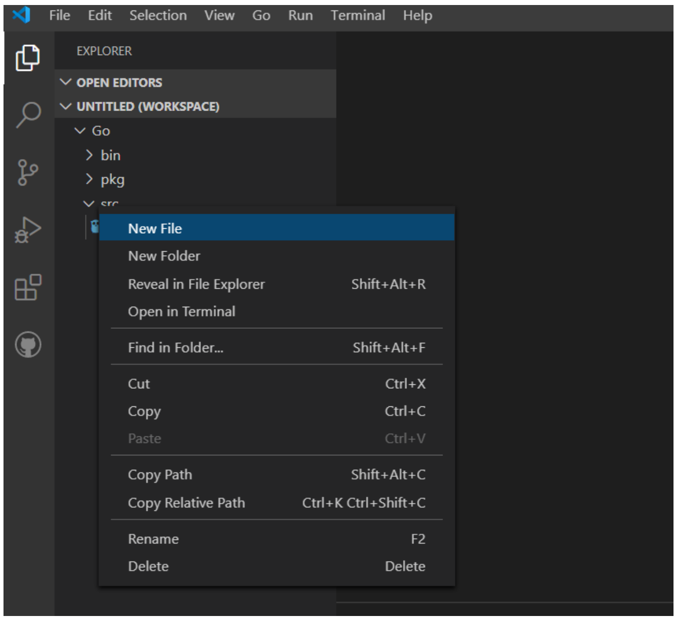
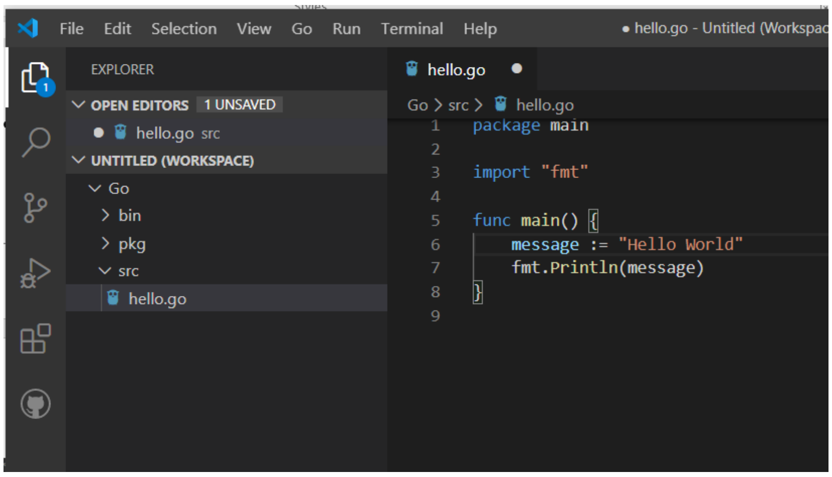


Figure 1. New File in Visual Studio Code



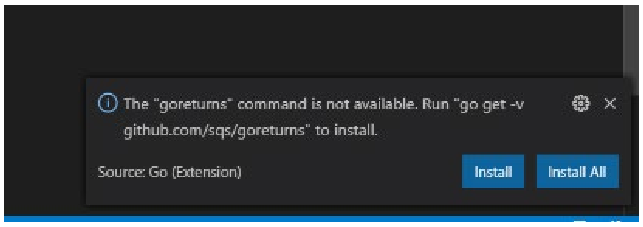
```go
package main

import "fmt"

func main() {
    message := "Hello World"
    fmt.Println(message)
}
```

Figure 2. Hello world program in Go

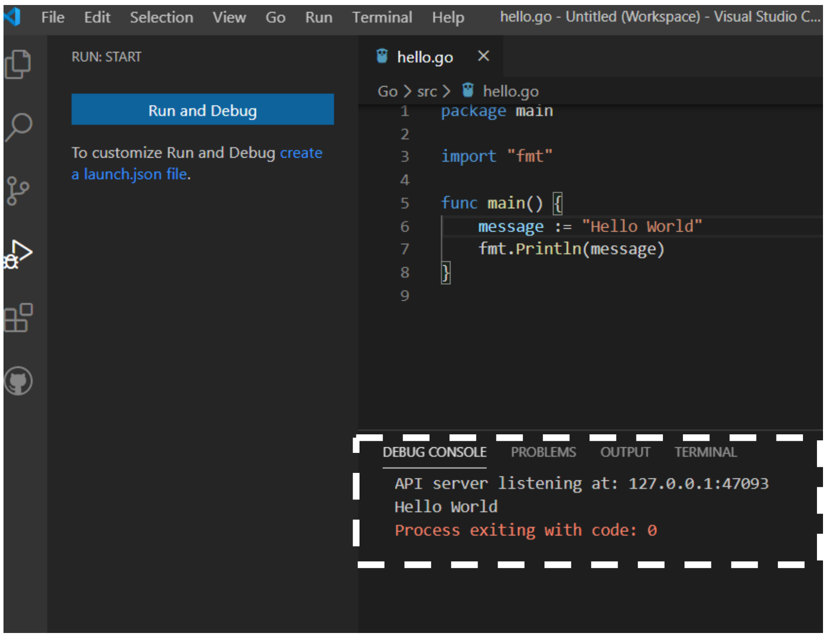
Now press Ctrl-F5 (run without debugging) and you may be prompted to install some missing packages. Go ahead with the installation.


The "goreturns" command is not available. Run "go get -v github.com/sqs/goreturns" to install.

Source: Go (Extension)

Figure 3. Prompt for installation of packages

After successful build and run, you will observe the output as follows:


File Edit Selection View Go Run Terminal Help
hello.go - Untitled (Workspace) - Visual Studio C...
RUN: START
Run and Debug
Go > src > hello.go
DEBUG CONSOLE
PROBLEMS
OUTPUT
TERMINAL

To customize Run and Debug create a launch.json file.

```go
package main

import "fmt"

func main() {
    message := "Hello World"
    fmt.Println(message)
}
```

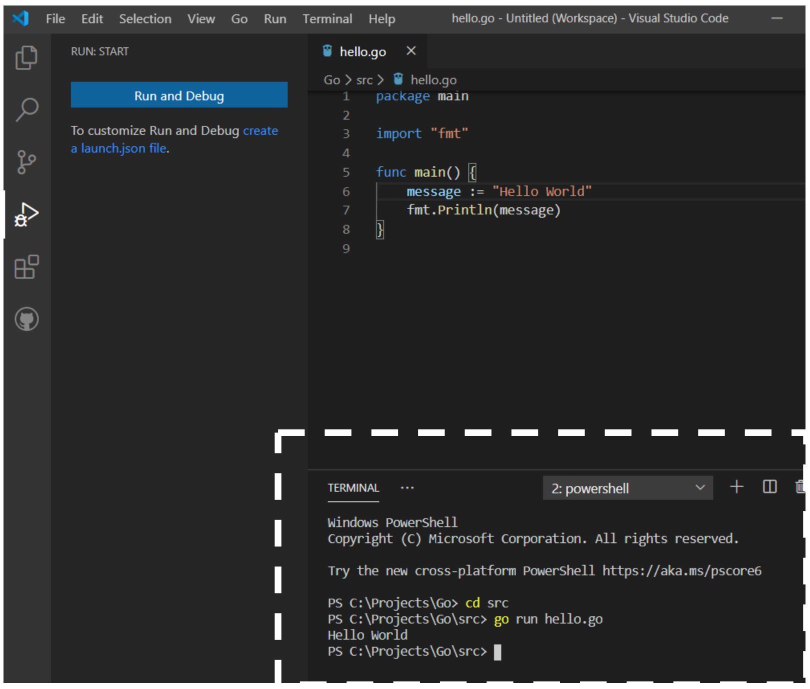
```
API server listening at: 127.0.0.1:47093
Hello World
Process exiting with code: 0
```

Figure 4. Program output

You can also run the code from the console

```
>>cd %GOPATH%\src
>>go run hello.go
Hello World
```

Or you may use the integrated command line terminal in Visual Studio Code via "Terminal" > "New Terminal".


File Edit Selection View Go Run Terminal Help
hello.go - Untitled (Workspace) - Visual Studio Code
RUN: START
Run and Debug
To customize Run and Debug create a launch.json file.
hello.go
Go > src > hello.go
TERMINAL
2: powershell

```go
package main

import "fmt"

func main() {
    message := "Hello World"
    fmt.Println(message)
}
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Projects\Go> cd src
PS C:\Projects\Go\src> go run hello.go
Hello World
PS C:\Projects\Go\src>
```

Figure 5. Integrated Command Line Terminal

	go run <*filename*>	**Go command for running Go program file.**
	import	**Keyword use for bringing in packages.**
	fmt	**Package for formatting and reading text.**
	main()	**Program entry point.**
	//	**For single-line comments.**
	/* … … */	**For spanning multiple lines comments.**
⚠	**Go operates in packages instead of files.** **No conflict between name of package and name of function.** **Go packages may have init() functions executed before main().** **Go programs do not use semicolons but are added in by compiler, but will need to be used if multiple statements are on the same line.**	

NGEE ANN POLYTECHNIC

Basic Data Types

Keywords, Variables and Constants

Description	*In this lesson, you will learn the keywords, variables and constants in Go.*
Learning Objectives	• Know the keywords in Go • How to declare variables • How to assign variables
Duration	40 minutes

Keywords

A Go identifier is a nonempty sequence of letters and digits. The Go compiler prohibits the use of identifiers that have the same name as keywords. Go also has many predefined identifiers and these predefined identifiers are not reserved, so it is possible to redeclare them. However this needs to be done with caution as there is potential for confusion.

Table of Keywords

break	**default**	**func**	**interface**	**select**
case	**defer**	**go**	**map**	**struct**
chan	**else**	**goto**	**package**	**switch**
const	**fallthrough**	**if**	**range**	**type**
continue	**for**	**import**	**return**	**var**

Table of Predefined Identifiers

append	**copy**	**int8**	**nil**	**true**
bool	**delete**	**int16**	**panic**	**uint**
byte	**error**	**int32**	**print**	**uint8**
cap	**false**	**int64**	**println**	**uint16**
close	**float32**	**iota**	**real**	**uint32**
complex	**float64**	**len**	**recover**	**uint64**
complex64	**imag**	**make**	**rune**	**uintptr**
complex128	**int**	**new**	**string**	

Go also allows for the use of a blank identifier ("_"). Blank identifier used as placeholder to receive expected variable and discards it when given. Some examples are given below.

```
weight, error = fmt.GetWeight(10)        // Get weight and error code
weight, _ = fmt.GetWeight(10)            // Get weight; discard error code
_, error = fmt.GetWeight(10)             // Discard get weight; get error code
```

	letters	**The underscore "_" or any character that is in the Unicode categories "Lu, Ll, Lt, Lm and Lo" and including A-Z and a-z.**
	digit	**Any character in the Unicode category "Nd" and including Arabic digits (0-9)**
	scope	**Identifiers starting with capital letters are considered public i.e. visible and accessible outside of their own package; those that are not are private, so visible only to files of their own package.**
	package **camel case**	**Package names by default are always in lower case.** **Go identifiers are usually named in "camel case", ie. interior capital letters, e.g. QuoteRuneToASCII or parseRequestLine.**
⚠	**identifiers**	**Case-sensitive.** **InfoGiven, Infogiven, infogiven, infoGivenand INFOGIVEN are all different identifiers.**

Variables and Constants

Variables are declared using **var** keyword or using short variable declaration (:=) syntax. Constants are declared with the **const** keyword.

```
date := 24                  // variable; inferred type: int
end := int64(9876543210)    // variable; type: int64
var age int                 // variable; value 0; type: int
const population = 65530    // constant; type: uint16
var age, weight int = 12, 20 //variable; type: int

var (
       name string
       age, location int
       country = "Singapore"
)

const(
       first = 0
       second = 1
)
```

Code Snippet:

```
//type.go

package main

import "fmt"


func main(){
date := 24
const gravity = 10
var age int
fmt.Printf("%T %T %T\n", date, gravity, age)
}
```

Running in console (from now onwards, we will assume that you will first go into the appropriate directory containing the go source using cd command)

```
>>go run type.go
int int int
```

	%T	**"%T" is used as a Go-syntax representation of the type of the value.**

Activity #1: Simple Types Printout Program in Go

Create a Go application that has the following features.

1. Create a basic Go program using the standard template
2. Declare the following variables

- text of “The following is the account information.”
- first name “Luke”
- last name “Skywalkter”
- age of 20 yrs old
- weight of 73.0 kg
- height of 1.72 m
- remaining credits of $123.55
- account name of “admin”
- account password of “password”
- subscribed user as “true”

3. Use the package fmt to print out the values and types of the variables declared in part 2.

NGEE ANN POLYTECHNIC

Boolean values

Description	***In this lesson, you will learn Boolean values and the use of Boolean expressions.***
Learning Objectives	• **Know the different general uses of Boolean values** • **Understand the use of expressions**
Duration	**30 minutes**

Booleans

Boolean in Go is provided as true or false with a type bool. Go supports standard logical and comparison operators which are always returned as a bool result.

Table of boolean conditions

Condition 1	Condition 2	Condition 1 && Condition 2	Condition 1 || Condition 2	Not Condition 1
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Table of comparison operators

Syntax	Result
!A	Logical NOT operator - True if the expression A is False.
A || B	True if either condition is True.
A && B	True if both conditions are True.
A < B	True if A is less than B, False otherwise.
A <= B	True if A is less than or equal to B, False otherwise.
A == B	True if A is equal to B, False otherwise.
A != B	True if A is not equal to B, False otherwise.

Code Snippet:

```
//boolean.go

package main

import "fmt"


func main(){
	conditionA := 10
	conditionB := 12
	conditionC := 10

	if conditionA < conditionB {
		fmt.Println("A lesser than B")
	}
	if conditionB <= conditionC {
		fmt.Println("A lesser than C")
	}
	if conditionA == conditionC {
		fmt.Println("A is same as C")
	}
}
```

Running in console:

```
>>go run boolean.go
A lesser than B
A lesser than C
A is same as C
```

Activity #1: Simple True-False Program

Create a Go application that has the following features.

1. Request the user to input a value between 1 to 100.
2. Compare the value against a predetermined value and determine an output.if

- If it is correct, print “Well Done! Your guess is correct”.
- If it is too low, print “Too low, try again next time!”.
- if it is too high, print “Too high, try again next time!”.

3. You may wish to use the following code snippet to read in the user input for comparison.

Code Snippet:

```
//UserInput.go

package main

import(
       “fmt”
)

func main(){
var guess int
fmt.Println("Enter an integer value: ")
fmt.Scanln(&guess)

}
```

Numerical Types

Description	*In this lesson, you will learn the different numerical types*
Learning Objectives	• Do basic arithmetic formulae using the basic math • Do advanced formulae using "Math" package
Duration	90 minutes

Integer, Floating and Complex Types

Go has various data types for numerical values.
The following types are for integers and their respective ranges.

int8	-128 to 127
int16	-32768 to 32767
int32	-2147483648 to 2147483647
int64	-9223372036854775808 to 9223372036854775807
byte	uint8
uint8	0 to 255
uint16	0 to 65535
uint32	0, 4294967295
uint64	0, 18446744073709551615
uintptr	unsigned integer storing pointer value

The following types are for integers and their respective ranges.

float32	$\pm 3.40282346638528859811704183484516925440 \times 10^{38}$
float64	$\pm 1.797693134862315708145274237317043567981 \times 10^{308}$
complex64	The real and imaginary parts of type float32
complex128	The real and imaginary parts of type float64

NGEE ANN POLYTECHNIC

Activity #1: Simple Calculator Program in Go - Basic calculation

Create a Go application that has the following features.

1. Request the user to input a value from 0, 1, 2, 3 9
2. Request the user to select arithmetic functions add, subtract, multiply or divide.
3. Request the user to input a second value from 0,1,2,3...9
4. Print out the final result.

Activity #2: Temperature Converter - Kelvin to Degrees to Fahrenheit

Create a Go application that has the following features.

1. Request the user what format the input temperature is in.

(1 for Kelvin, 2 for Celsius and 3 for Fahrenheit.)

2. Request the user to input the current temperature.
3. Print out the final result showing both conversions to 2 decimal places.

The conversion given Fahrenheit (F) to Celsius (C) and Kelvin (K) is given as follows.

$C = \frac{5}{9}(F - 32)$	$K = \frac{5}{9}(F + 459.67)$

Activity #3: Currency calculator

Create a Go application that has the following features.

1. Allow users to enter the number of 1-dollar coins, 50-cent coins, 20-cent coins, 10-cent coins and 5-cent coins.
2. Calculate and print the total amount.
3. Calculate how many 2-dollar notes can be given in exchange and the remaining change to return.

You may wish to use the following code snippet to read in the user input for this activity.

Code Snippet:

```
//UserInput.go

package main

import(
        "fmt"
)

func main(){
var inputValue int
fmt.Println("Enter an integer value: ")
fmt.Scanln(&inputValue)

}
```

Activity #4: Triangle side calculator

Create a simple triangle side calculator.

1. Allow the user to input 2 lengths of the triangle and an angle.
2. Calculate the resultant length using the “Math” package and the cosine rule.
3. Print the resultant length.

$$a^2 = b^2 + c^2 - 2bc \cos A$$

NGEE ANN POLYTECHNIC

Strings

Description	***In this lesson, you will learn the manipulation of Strings***
Learning Objectives	• **Know the general use of String methods.** • **Basic understanding in manipulating Strings.**
Duration	**90 minutes**

String Manipulation

Go contains various options to manipulate String data.

Syntax	Description
s == r	Compare String s and String r.
s+=t	appends String t to the end of String s.
s + t	Concatenation of String s and t.
s[n]	Take from the index position n in String s.
s[n:m]	Takes from String s starting from index n to index (m-1)
s[n:]	Takes from String s starting from n to len(s) -1.
s[:m]	Takes from String s from index 0 to m-1.
len(s)	The number of bytes in String s.
String(i)	Converts i into String.

Activity #1: Simple Concatenation

1. Given the following
a. list1 := [4] string{“ans”, “wer”, “is”, “of”}
b. list2 :=[4] string{“-”, “+”, “*”, “/”}
c. list3 :=[4]string{“5”, “10”, “4”, “0”}

Construct and display the following statement using string manipulation techniques

Answer of 5 + 4 is 9

NGEE ANN POLYTECHNIC

Control Structure

IF STATEMENTS

Description	*In this lesson, you will learn the use of IF in Go.*
Learning Objectives	• Know the basic implementations of if statements • Know the basic implementations of if...else if statements • Know the basic implementations of if... else if ... else
Duration	15 minutes

IF statements

Go conditional statement executes the respective branch that is true.

ⓘ	**if**	**if conditional**
	if ... else if	**if conditional else alternative**
	if ... else if ... else	**if conditional else alternative else alternative**
	if i:=0; condition	**if statements can be preceded by a simple statement.**
⚠	**There is no ternary expression in Go. i.e. if conditional A ? conditional B { code }**	

Activity #1: If statements

using "if" statements, determine the following and print out the answer for each query.

Given the numbers is 17 and 24 and using preceding conditionals for at least 1 of the numbers.

1. Which number is bigger?
2. By how much is it bigger by?

Activity #2: If ... else if statements

using "if... else if" statements, determine the following and print out the answer for each query.

Given the number is 17 and using preceding conditionals,

1. Is the number an odd or even number?
2. Does the number have more than 1 digit?

Activity #3: If ... else statements

Write a program that prompts the user to enter an integer and outputs whether the number is divisible by both 5 and 6.

Activity #4: If … else if… else statements

using "if… else if … else" statements, determine the following and print out the answer for each query.

1. Given authenticated users have the follow usernames
a. Admin
b. Robin
c. John
2. Request for the user to input a name.
3. Print out "Welcome, Admin!" if the name is "Admin"
4. Print out "Welcome!" if the name is an authenticated user
5. Print out "Merry Men" if the name is not.

Activity #5: Compound If...else statements

A year is said to be leap year if it is divisible by 4 but not divisible by 100 except those that are divisible by 400. Write a program to determine whether an input year is a leap year or not.

NGEE ANN POLYTECHNIC

FOR STATEMENTS

Description	*In this lesson, you will learn the use of FOR in Go.*
Learning Objectives	• Know the basic implementations of For statements • Know the basic implementations of infinite For statements
Duration	15 minutes

FOR statements
Go uses FOR to unite "for" and "while" loops.

ⓘ	single	single condition to trigger for loop
	'For' clause	Always evaluated as true - while loop
	range clause	for init; conditional; post statement
⚠	Be careful of infinite loops, always ensure there is an exit condition unless specified not to.	

Activity #1: Numbers printing
1. Print numbers from 0 to 1000.
2. Print only the odd numbers from 0 to 1000.
3. Print only the even numbers from 0 to 1000.

Activity #2: Odd or even numbers
Create an application that always asks for a number from the user.
1. The number is then decided to be either odd or even and printed to the user.

Activity #3: For range
Create an application that asks the user for 2 values.
1. The values are stored as variables.
2. Print the odd numbers within range specified by the user.
3. Print the even numbers within the range specified by the user.

Activity #4: Infinite query For
Create an application that asks the user to input a number infinitely.
Given the number and using preceding conditionals, determine
1. Is the number an odd or even number?
2. Does the number have more than 1 digit?

Activity #5: Up and Down counter
Create an application that asks the user to input 2 values.
1. The application will then count up from the starting number to the ending number and count back down to the starting number from the ending number in intervals of 1 step.

Activity #6: Number guessing game

Create a simple application that simulates a number guessing game. It first generates a random number between 1 and 100. It then prompts the user to guess the correct number. The user can enter 101 to end the game or the game will end after 5 tries.

You may wish to use the following code snippet to read in the user input for this activity.

Code Snippet:
```
//UserInput.go

package main

import(
       "fmt"
       "bufio"
       "os"
)

func main(){
       reader := bufio.NewReader(os.Stdin)
       fmt.Print("Enter Name: ")
       text, _ := reader.ReadString('\n')
       fmt.Println(text)
}
```

You may wish to use the following code snippet to generate a random number.

Code Snippet:
```
//UserInput.go

package main

import(
       "fmt"
       "math\rand"
       "time"
)

func main(){
rand.Seed(time.Now().UnixNano())
x:= rand.Intn(5)// an int between 0 to 5
}
```

NGEE ANN POLYTECHNIC

Activity #7: Verify Vehicle Plate Number Authenticity

1. Request the user to enter a vehicle plate number.
2. With the input do the following,
a. For a 3 letter prefix, only the last 2 letters are used.
b. For a 2 letter prefix, both letters are used.
c. For a 1 letter prefix, the letter corresponds to the second position. The first position is defaulted to 0.
d. Both letters are converted into numbers with A = 1 and Z = 26.
e. For numerals less than 4 digits, additional zeros are added in from as placeholders. i.e. “1” is “0001”. An input of SBS3229 would therefore give 2, 19, 3, 2, 2 and 9; E12 would give 0, 5 , 0 , 0 , 1 and 2.
f. Each individual number is to be multiplied to 6 fixed numbers (9, 4, 5, 4, 3, 2).
g. The resultant is added up and divided by 19.
h. The remainder corresponds to one of the 19 letters in the checksum given below with “A” corresponding to remainder of 0, “Z” corresponding to 1, “ Y” corresponding to 2 and so on.
3. Based on the last letter given by the user vehicle plate number,
a. Print “The vehicle plate given is correct!” if the checksum matches.
b. Print “The vehicle plate given is not correct!” if the checksum does not matches.

<u>Checksum letters</u>

(A, Z, Y, X, U, T, S, R, P, M, L, K, J, H, G, E, D, C, B)

You may wish to use the following code snippet to read in the user input for this activity.

Code Snippet:
```
//UserInput.go

package main

import(
       "fmt"
)

func main(){
var inputValue int
fmt.Println("Enter an integer value: ")
fmt.Scanln(&inputValue)

}
```

SWITCH STATEMENTS

Description	***In this lesson, you will learn the use of Switch in Go.***
Learning Objectives	• **Know the basic implementations of Switch statements** • **Understand the use of fallthrough**
Duration	**15 minutes**

Switch statements

Switch is an alternative to if statements.

ⓘ	**Switch, Case, Default**	**the typical style of switch in Go.**
	Switch cases	**Need not be constant or integers.**
	Multiple cases	**switch can exist with multiple cases.**
⚠	**Fallthrough is not default in Go. Break is not needed.**	

Activity #1: Day of the week

Create a switch case that determines the day of the week and prints out the corresponding day.
You may use the following syntax to extract the data of the week from the system

Include : time
Syntax: time.Now().Weekday()

Determine if the day is odd or even.

Activity #2: BMI

Create a switch case that determines the BMI of the user.

1. Request the user to input their height and weight.
2. Calculate the BMI based on the input using the BMI formula below:

$$BMI = \frac{Weight\ (kg)}{[Height(m)]^2}$$

3. Advise the user on the BMI standards based on the table below:

Weight Categories	BMI (kg/ m^2)
Underweight	<18.5
Healthy Weight	18.5 to 24.9
Overweight	25 to 29.9
Obese	30 to 34.9
Severely Obese	35 to 39.9
Morbidly Obese	>=40

Activity #3: Fallthrough

Evaluate the output of for each of the switches below:

```
switch {
case 20 > 40:
        fmt.Println("20 > 40")
case 10 > 1:
        fallthrough
case 1 > 10:
        fmt.Println("1 > 10")
default:
        fmt.Println("None 1")
}

switch {
case 10 > 11:
        fmt.Println("10 > 11")
case 5 > 1:
        // fallthrough
case 1 > 10:
        fmt.Println("1 > 10")
default:
        fmt.Println("None 2")
}
```

```
switch {
case 20 > 21:
	fmt.Println("20 > 21")
case 10 > 1:
	break
case 20 > 10:
	fmt.Println("20 > 10")
default:
	fmt.Println("None")
}
```

Composite Types

Arrays

Description	*In this lesson, you will learn the use of arrays in Go.*
Learning Objectives	• Know the basic ways to create arrays. • Know how to do simple access to arrays. • Know how to add new elements to arrays. • Know how to slice arrays.
Duration	15 minutes

Arrays

Arrays serve as primary building blocks for slices in Go.

ⓘ	**Mapping of elements**	**It is in integer**
	Indexes	**Non-negative**
	Base index	**Zero based**
⚠	**It is idiomatic in Go to use Slices instead of Arrays.**	

Activity #1: Declare array variable with a specific size

Declare a basic array for the following data and store as string.

Operating System List
1. Windows XP
2. Linux 1.0
3. Raspbian Teensy
4. MacOS
5. IOS
6. Google Android

Print out the length of each element.

Activity #2: Assess a particular element in an array and update the element

Make use of the array in Activity #1 and assign new elements .
1. Windows XP -> Windows 10
2. Linux 1.0 -> Linux 16.04
3. Raspbian Teensy -> Raspbian Buster

Print out the updated array

Activity #3: Add elements to an array

Make use of the array in activity #2 and add the following to the array.
1. Ubuntu
2. MS-Dos
3. Solaris

Print out the updated array

Activity #4: Assess a particular segment in an array

1. Make use of the array in activity #3 and print out the following:

a. first 3 elements
b. the next 3 elements
c. the last 3 elements

Activity #5: Temperature Sensor

A room is installed with a sensor that measures the temperature at an hourly interval. Create an array that stores the measurements

24 hr measurements -

20.1	24	27.3	30.1	26.4	22.2	20.1	24	27.3	30.1
26.4	20.1	24	27.3	30.1	26.4	20.1	24	27.3	30.1
26.4	20.1	24	27.3						

Using a for loop,
Calculate the average temperature for the day.

Slices

Description	***In this lesson, you will learn the use of slices***
Learning Objectives	• **Know the basics of using slices** • **Know how to access the element of the slice** • **Know the difference in slice length and slice capacity** • **Know how to do subslicing**
Duration	**20 minutes**

Slices

Slices are more powerful, flexible and convenient than an array. It is a lightweight data structure.

ⓘ	**Length**	**The length of the elements in the slice**
	capacity	**The maximum size upto that it can expand**
	over capacity	**The capacity is reconstructed if the elements are more than the original capacity. By default, it is doubled in capacity.**
⚠	**The first index is always 0 and the last is always (length of slice -1)**	

Activity #1 Declare a new slice

Create a slice for this month's spendings with a capacity of 5
Week 1 - $9.50
Week 2 - $8.00
Week 3 - $10.20
Week 4 - $7.40
Print out the length and capacity of the slice.

Activity #2 Access the element of the slice

Use the slice declared in activity #1.Print out the spending for week 3.
Change the spending for week 3 to $9.80

Print out the new slice.

Activity #3 Capacity after append

Use the slice declared in activity #2. Append the following
1. Week 5 - $8.40
2. Week 6 - $9.40
3. Week 7 - $7.20

Print out the length and capacity of the slice.

Activity #4 Slicing

Use the slice declared in activity #3.
Sub slice the slice from position 3 to the end.

Print out the length and capacity of the slice.

Activity #5: Temperature Sensor with nested loop and slices

A flat unit has 3 rooms. Each room is installed with a sensor that measures the temperature of the room at random intervals and adjusts the air conditioning correspondingly.
Create a slice in a slice with the following data

Calculate the average temperature for each room using a nested loop with a nested slice.

Room 1	20	21	23	25	22	
Room 2	27	23	25	20	30	24
Room 3	22	23	24	22		

NGEE ANN POLYTECHNIC

Struct

Description	***In this lesson, you will learn basic construction of Struct and object creation using struct.***
Learning Objectives	• **Know the basics of Struct.** • **To be able to create objects based on Struct**
Duration	**20 minutes**

Struct

A user-defined type that groups or combines items into a single type. items can be different types.

ⓘ	**Compact**	**Can be made compact by grouping the same type in declaration.**
	name: value	**This method of declaration can be used to initialize only the subset of fields.**
	dot notation	**The values can be accessed via the dot notation.**
⚠	**The field values need to be passed in the same order as they are declared in the struct.**	

Activity #1 Declare a struct for user

Create a struct for customer with the following parameters

1. First name , fname
2. Last name, lname
3. Age, int
4. Subscriber, boolean
5. HomeAddress, string
6. Phone, int
7. CreditAvailable, float32
8. CurrentCartCost, float32
9. CurrentOrderCost, float32

NGEE ANN POLYTECHNIC

Activity #2 Declare 2 users using struct

Create 2 customer profile as follows:

Customer 1:

1. First name - Annakin
2. Last name - Skywalker
3. Age - 45
4. Subscriber yes
5. HomeAddress - “Death Star”
6. Phone - “ 1234567”
7. CreditAvailable - “10,000.00”
8. CurrentCartCost, “0.00
9. CurrentOrderCost, “0.00”

Customer 2:

1. First name - Han
2. Last name - Solo
3. Age - 50
4. Subscriber no
5. HomeAddress - “Tatooine”
6. Phone - “ 4321765”
7. CreditAvailable - “20,000.00”
8. CurrentCartCost, “0.00
9. CurrentOrderCost, “0.00”

Print out the details of the two customers in a paragraph describing their profile.

Description	***In this lesson, you will learn Maps***
Learning Objectives	• **Know how to create maps** • **Know how to retrieve values from maps** • **Know how to combine struct with maps** • **Know how to create and access nested maps**
Duration	**20 minutes**

Maps

A versatile data that is widely used as it allows for fast lookups and retrieval, update and deletion of values with the use of a key.

	Reference	**It is similar to a hash table**
	AKA	**also known as hash map, hash table, dictionary.**
	Cost	**Inexpensive to use**
	A Key must always be unique and always comparable by using == operator or support != operator.	
	The types of keys and type of values must be the same type.	

Activity #1 Declare a currency conversion map

Create a currency conversion map using the following data

Currency	Currency Name	Excchange rate = 1 EUR
USD	US dollar	1.1318
JPY	Japanese yen	121.05
GBP	Pound sterling	0.90630
CNY	Chinese yuan renminbi	7.9944
SGD	Singapore dollar	1.5743
MYR	Malaysian ringgit	4.8390

Source:
https://www.iban.com/exchange-rates

You are required to store the currency, currency name and exchange rate using Struct with Map.

Retrieve and print any currency of choice, printing the currency, currency name and exchange rate as your output.

Activity #2 Currency conversion application

Request for a user to input the following information:

1. Currency to convert from
2. Amount of currency
3. Currency to convert to

Print out the resultant converted value to the user.

The formula for converting is:

$$\boxed{\text{(Amount of currency / Currency to convert from) * Currency to convert to}}$$

Description	***In this lesson, you will learn Boolean values and the use of expressions.***
Learning Objectives	• **Appreciate basics of channel in go.**
Duration	**20 minutes**

Channel

It is a medium where goroutines communication with one another.

ⓘ	**Bidirectional**	**It is bidirectional by default**
⚠	**It only allows transport of the same type of data on the same channel.**	

Activity #1 Decipher the Channel

Based on the code below and the basic understanding of channels. Determine the output of this code without running it.

```
package main

import (
"fmt"
)

func producer(channel chan int) {
for i := 0; i < 10; i++ {
        channel <- i
}
close(channel)
}

func main() {
ch := make(chan int)
go producer(ch)  // trigger a concurrent routine to happen and calls “producer” to start
for {
        v, ok := <-ch
        if ok == false {
                break
        }
        fmt.Println("Received ", v, ok)
}
}
```

Activity #2 Decipher the Channel

Based on the code below and the basic understanding of channels. Determine the output of this code without running it.

```
package main

import "fmt"

func main() {
ch := make(chan int, 2)
ch <- 1
ch <- 2

fmt.Println(<-ch)
fmt.Println(<-ch)
}
```

Overfill the input channel buffer and see what happens.