Lab 01

Introduction to Problem Solving & Flow of Control

Basic Programming via code.org

Problem Solving & Flow of Control

Computer Operation: consists of three main parts.

- **Input**
- **Process**
- **Output**

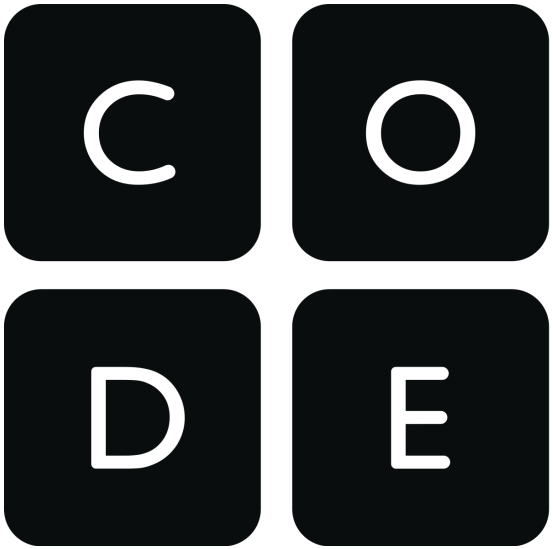
Problem Solving & Flow of Control

- **Processing Data:**
  - Accepts and gather data. (**INPUT**)
  - **Processes** data using problem solving methodology.
  - Achieve and present required result(s). (**OUTPUT**)
- **Flow of Control:** sequences/steps of program instructions used to solve a problem.
  - **Sequential**
  - **Selection or Conditional**
  - **Repetition or Loop**
  - **Invocation or Calling Function**

Flow of Control

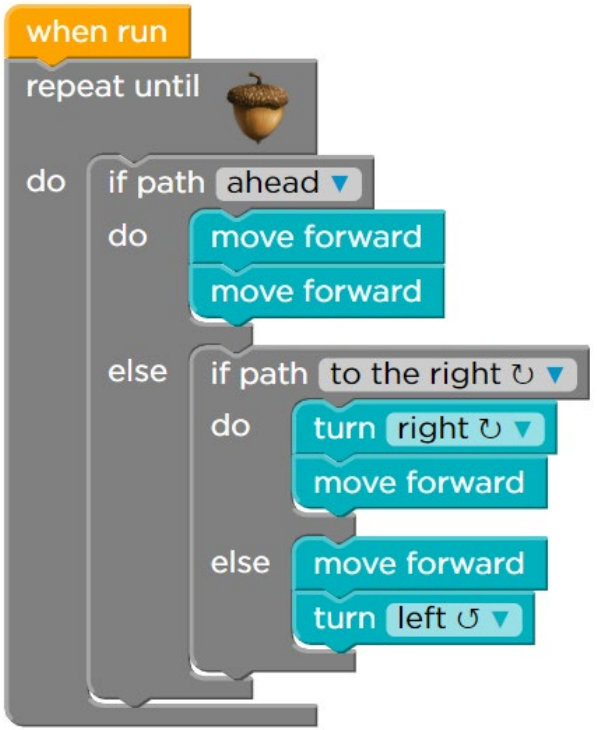
- **Sequential:** Normal flow of control for all programs.
- **Selection:** is used to select which statements are performed next based on a condition.
- **Repetition:** is used to repeat a set of statements.
- **Invocation:** is used to invoke a sequence of instructions using a single statement, as in calling a function

code.org



Introduction

- Today, we will work with one of easier approach to create a program – Block Programming
- Instead of typing up codes, you add instructions to the program by dragging and dropping blocks into the program

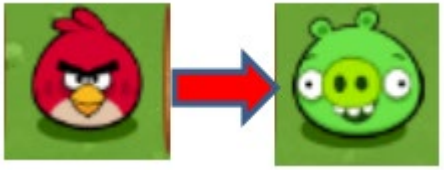
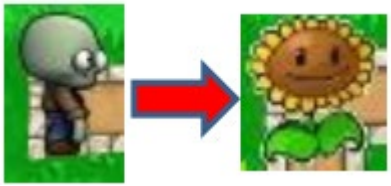


- In this lab we will work on a game called "**Classic Maze**" from **Hour of Code**.

- The main goal is to code different characters through the maze using available instructions.

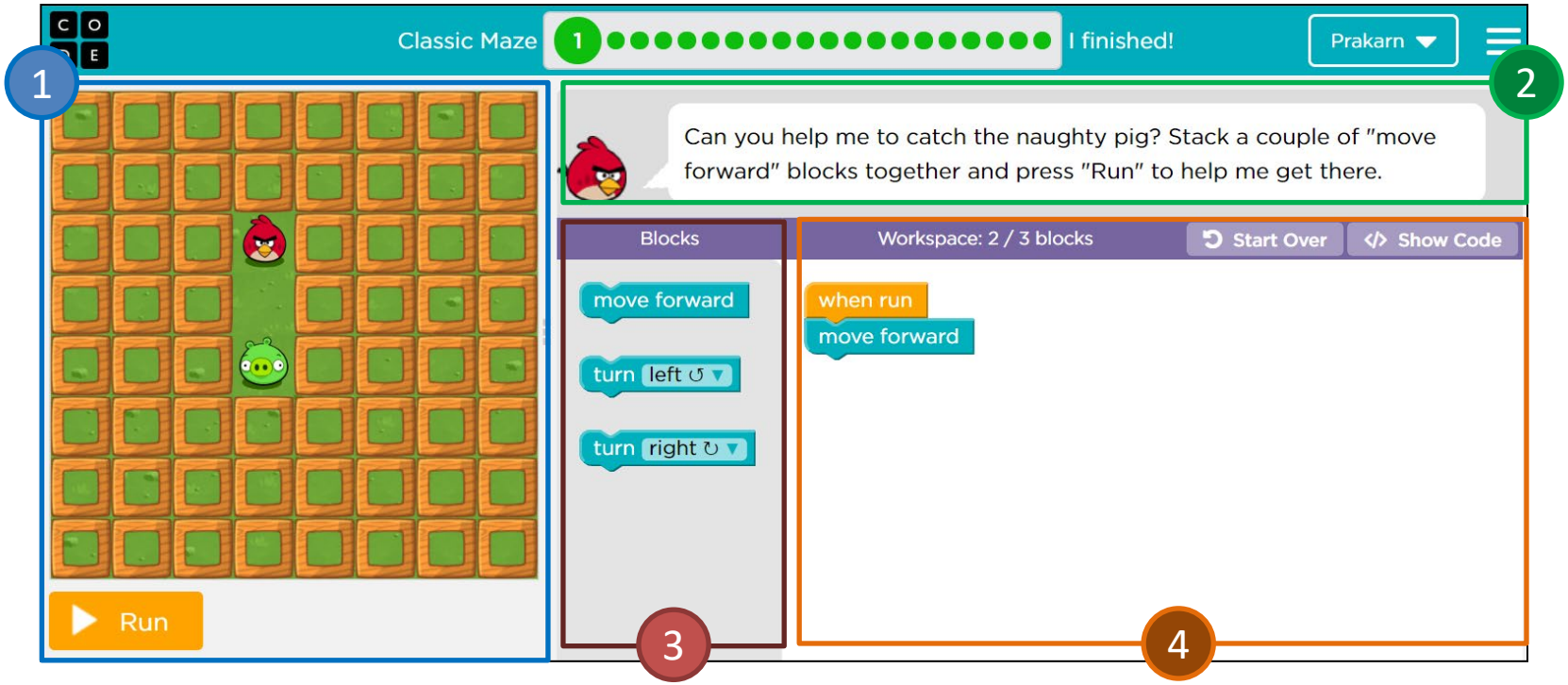
Objectives

- The goals of Classic Maze are:

1. Lead Angry Bird through the maze to the Green Pig, using available commands 
2. From level 12, you now will need to lead Zombie to the Sunflower 
3. From level 16, you will instead need to guide Scrat to the Acorn 

Note: Pay attention to the direction your character is facing!

What's on the Screen?



1. The Simulation

   (The result of your code)

2. The Instruction and objective

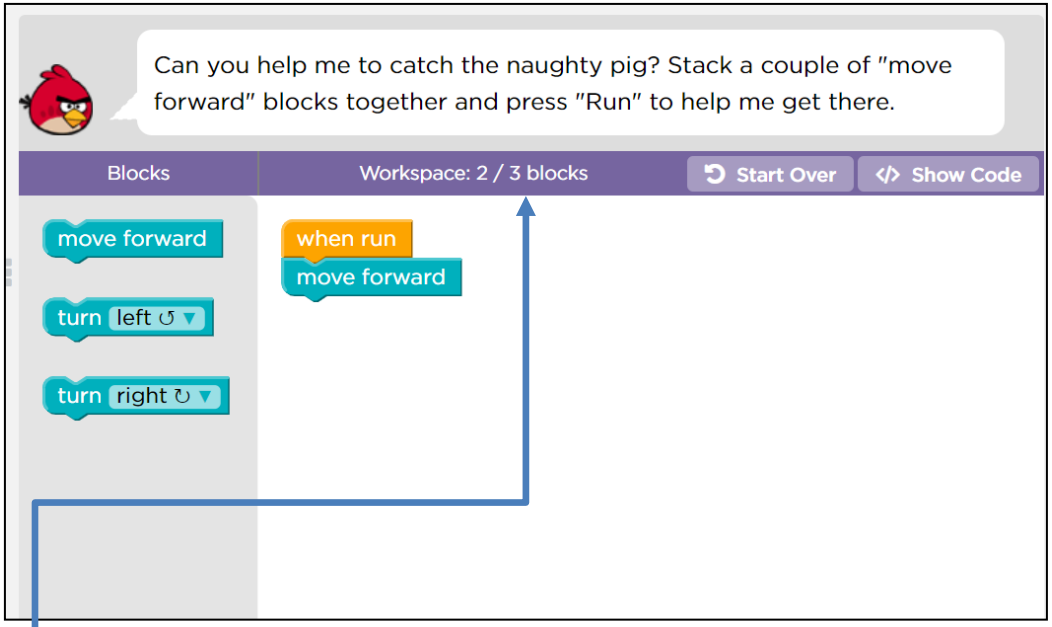
3. Available Blocks

   (Operations you can do)

4. Workspace

   (where you use operation)

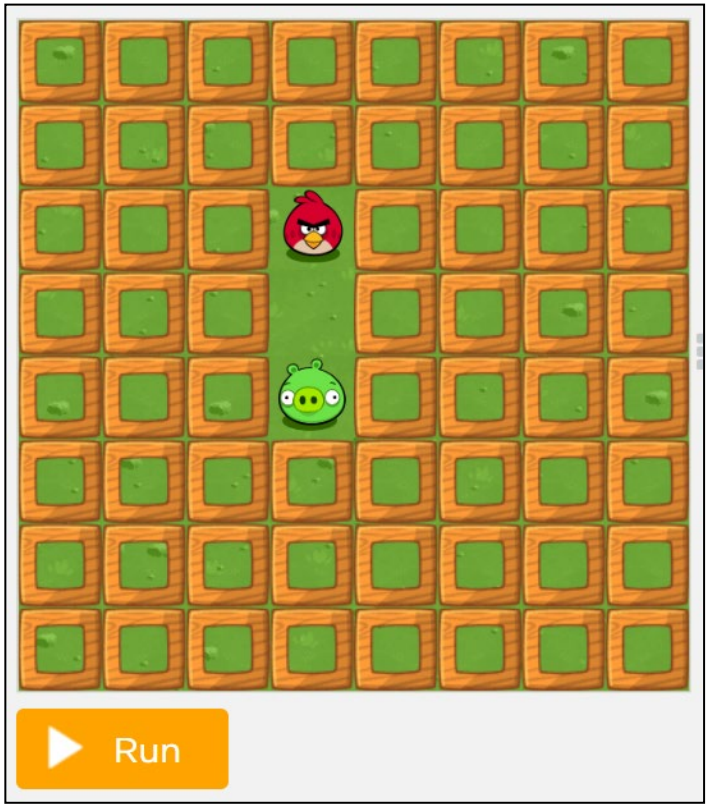
Right Portion – Objectives and Your Program



- To create a program, drag a block from the available blocks and add them to your program in the workspace.

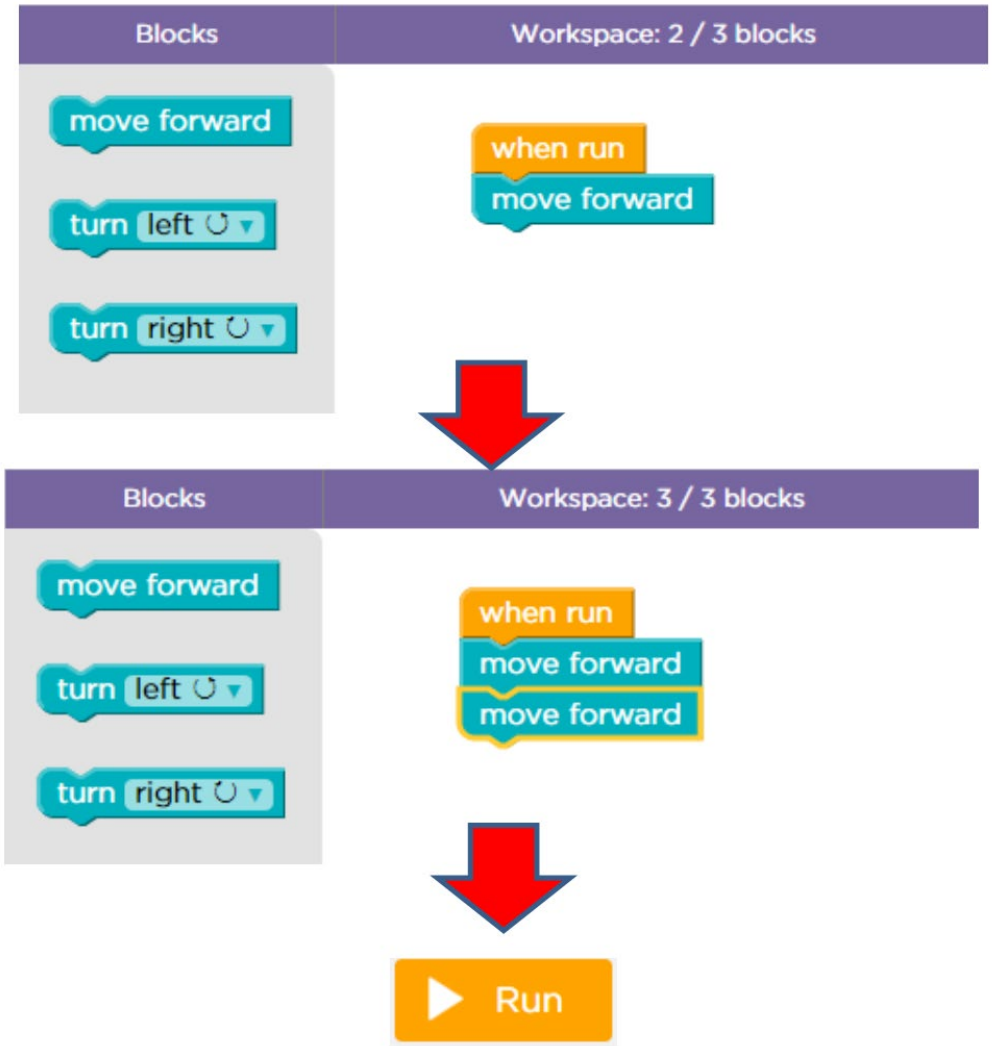
- When the program is run, it will start from “when run” block, the move down to the block below it (**sequential manner – from the top to the bottom**)
- You can use as many blocks of any type as you want, but the workspace will show the minimum number of blocks that can be use to finish the level
- “Start Over” button will reset your program
- “Show Code” will show the code of your program

Left Portion – The Simulation



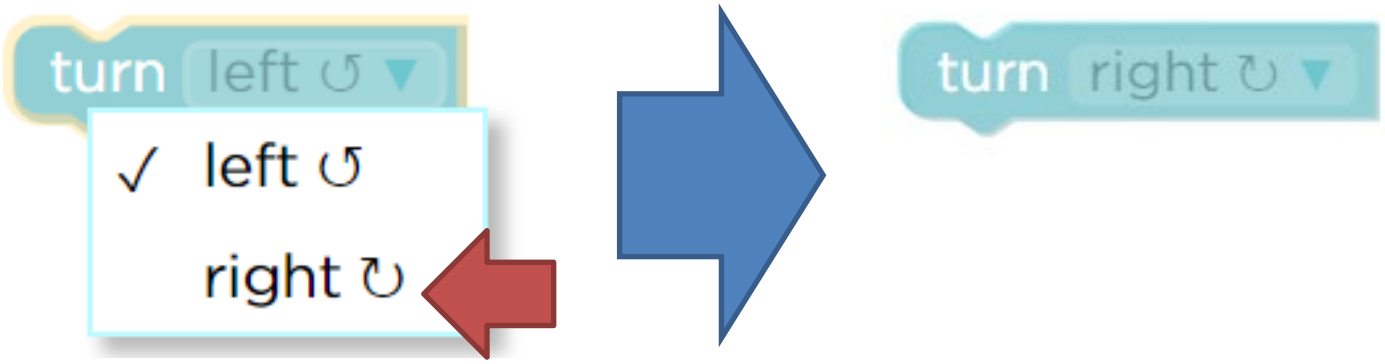
- Once your program is ready, click “Run” will run the program, showing the result on the maze
- While the program is running, the block currently running will be highlighted

Example – First level



Changing the Argument

- Some blocks, like turn, has a value you can change (left and right for turn). This is call **argument**, which will make the block behave differently



Loops

- Starting at Level 6, you will get a new block
  – "**repeat ... times**"

- "**repeat ... times**" block will repeat everything inside it for the number of times chosen as its argument.
- For example, the repeat block below will repeat "move forward" 5 times

You can put more than one block in here!

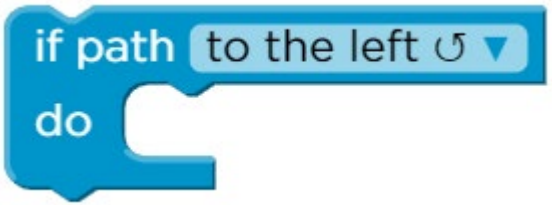
Loops: Repeat Until

- At level 10, you will get to use “**repeat until**”

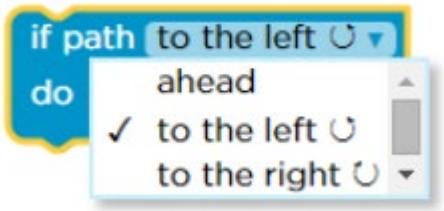
- “**repeat until**” will execute the code inside until the condition is met.
- In this case the block will repeat the code (can be more than one block) inside it until the pig is found.

If: Conditional

- At level 14, you will get an "**if**" block

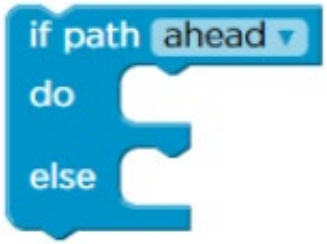


- "if" block will only execute the code inside it if the condition in the argument is true
  - It reads "**If there is a path to the left, do…**"
- As mention, if the condition is not what you want, you can change the condition by changing the argument



If-Else: Choosing between Two Choices

- From level 18, you will get the last block, "**if-else**"



- Similar to "**if**" block mentioned before, with addition "**else**" part.
- If the condition is true, the "**do**" part will be executed, if not, the "**else**" part will be executed

- For example above, the there is a path ahead, the character will move forward. Otherwise, the character will turn left.

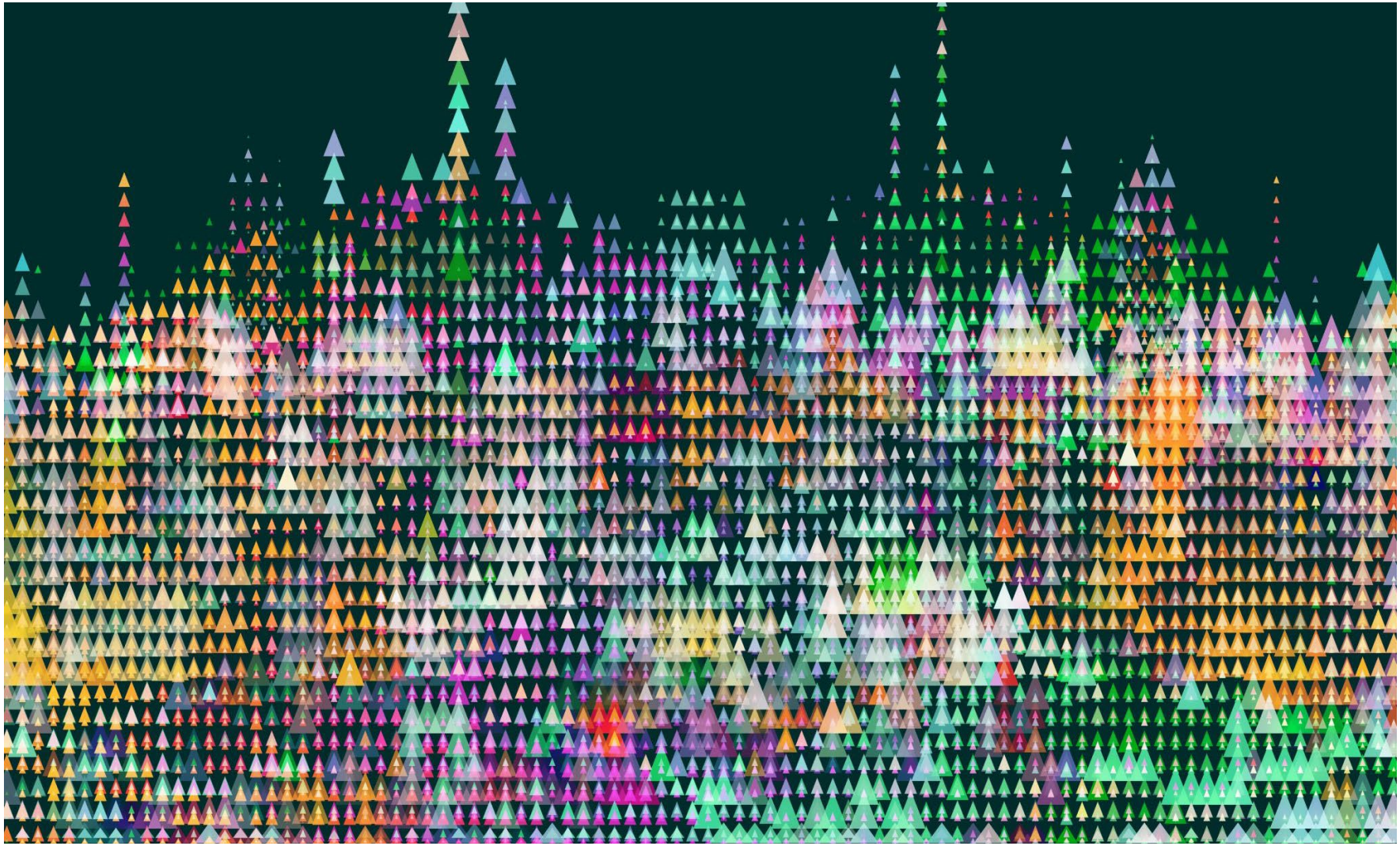
Nested Blocks

- Not only can you use as many block as you'd like, you can also put any inside another, creating more complex program

Let's Analyze This

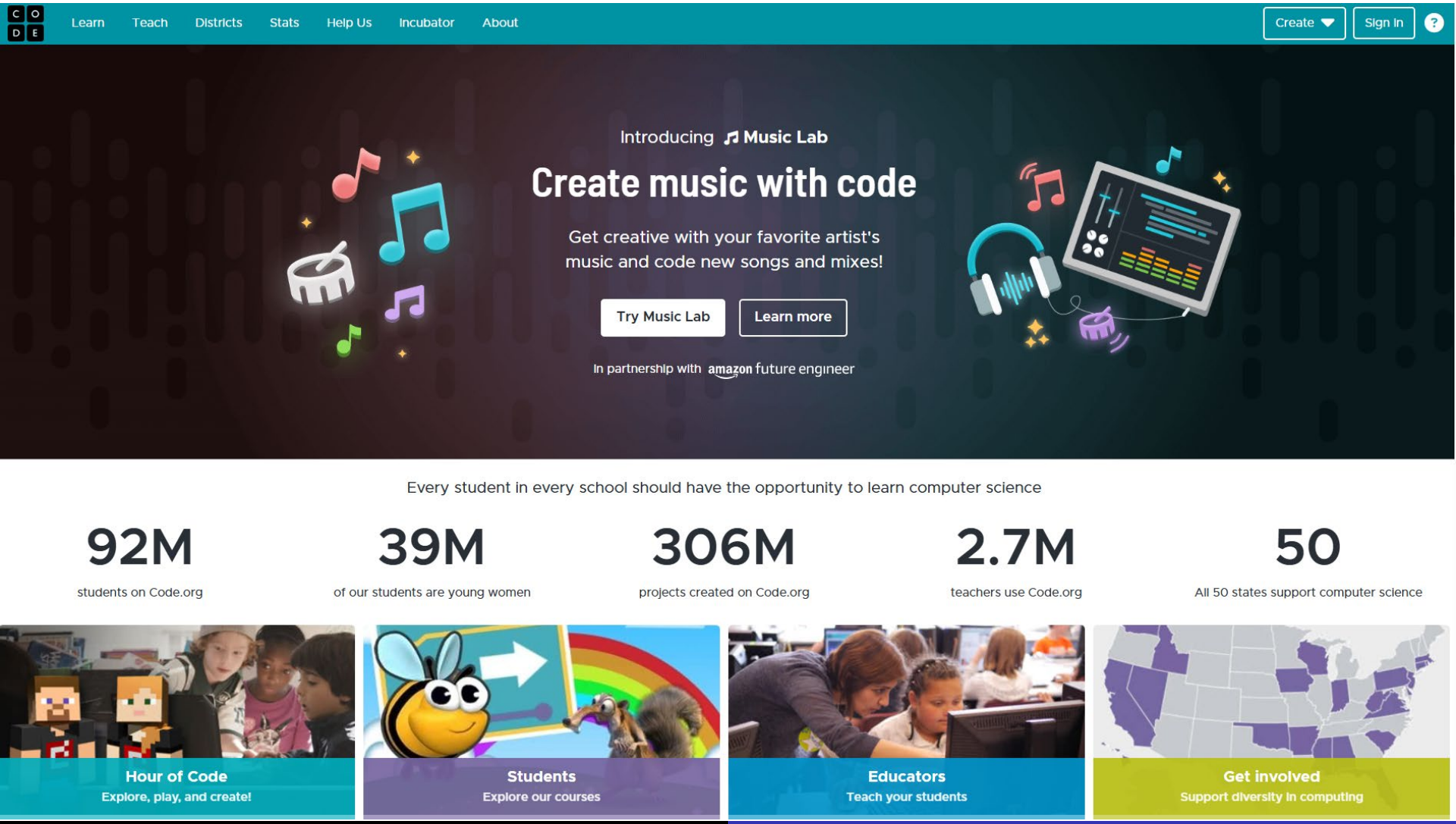
- Let's take a moment to analyze this:
- What is this supposed to do?
- The logic isn't entirely correct, can you tell me what is wrong with it?

Let Lab 1 begin



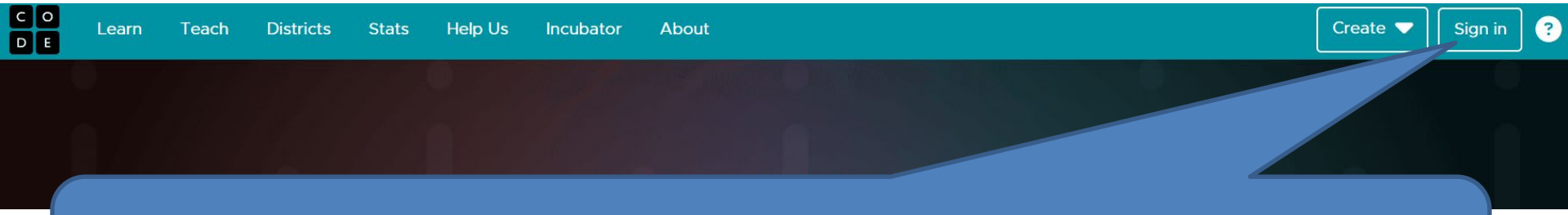
Accessing code.org

- From your web browser, go to https://code.org



Signing In

- If you are signed in on the code.org, your progress will be saved



- To sign in, select “Sign in” at the top-right corner of the page

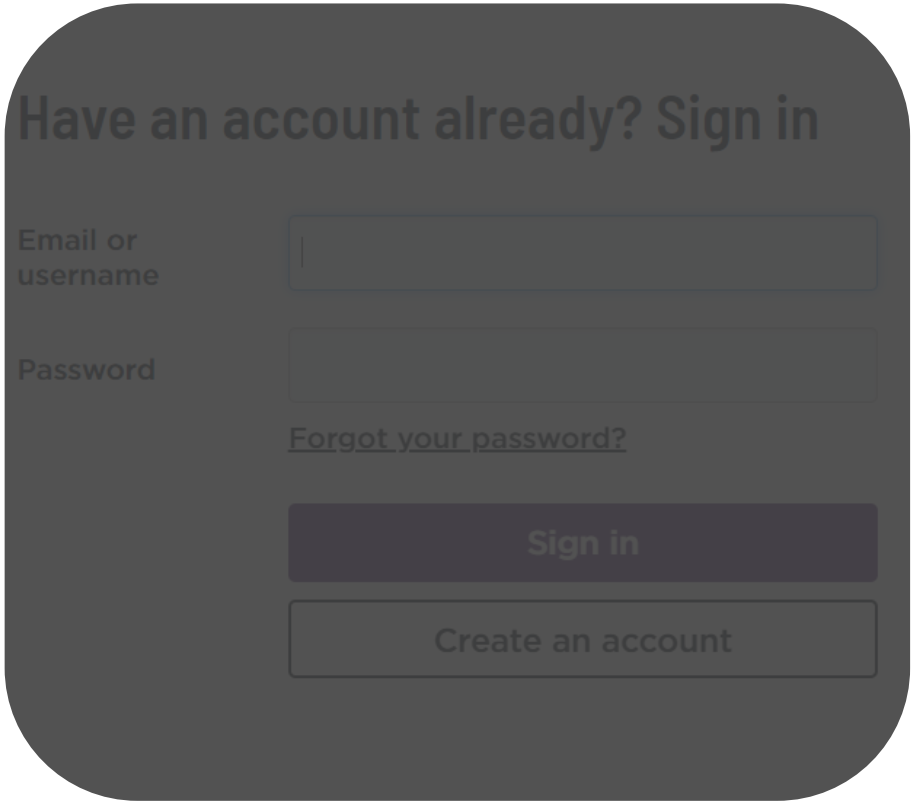
Signing In for the first time

input section code FIRST

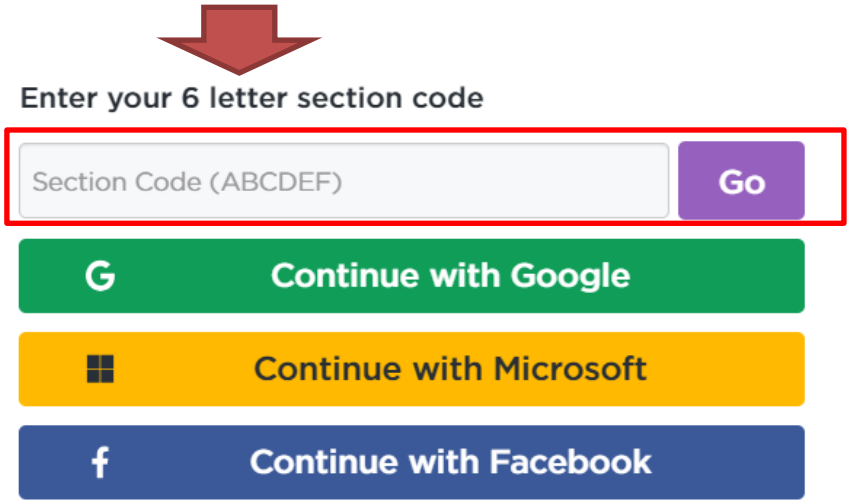
Sec701: **VSVJFF**

Sec702: **HRZKML**

Sec703: **QKQCJR**

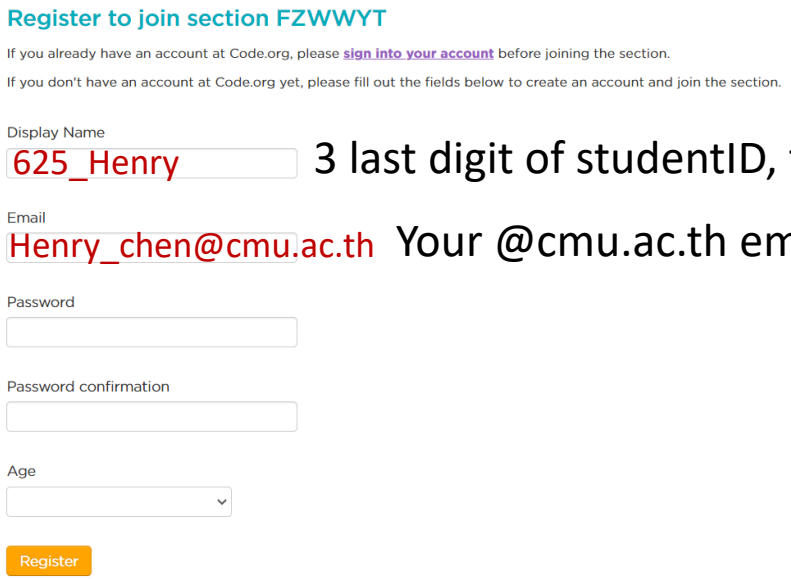


OR



Creating Account

- For the first time, you will then be asked to fill in the details.



3 last digit of studentID, followed by underscore and your first name

Your @cmu.ac.th email

Already join the class

You have successfully joined 1_66_204101_701.

Classic Maze

Try the basics of computer science with characters from Angry Birds, Plants vs. Zombies, and Scrat from Ice Age!

Try Now | Get Help | ✓ Assigned

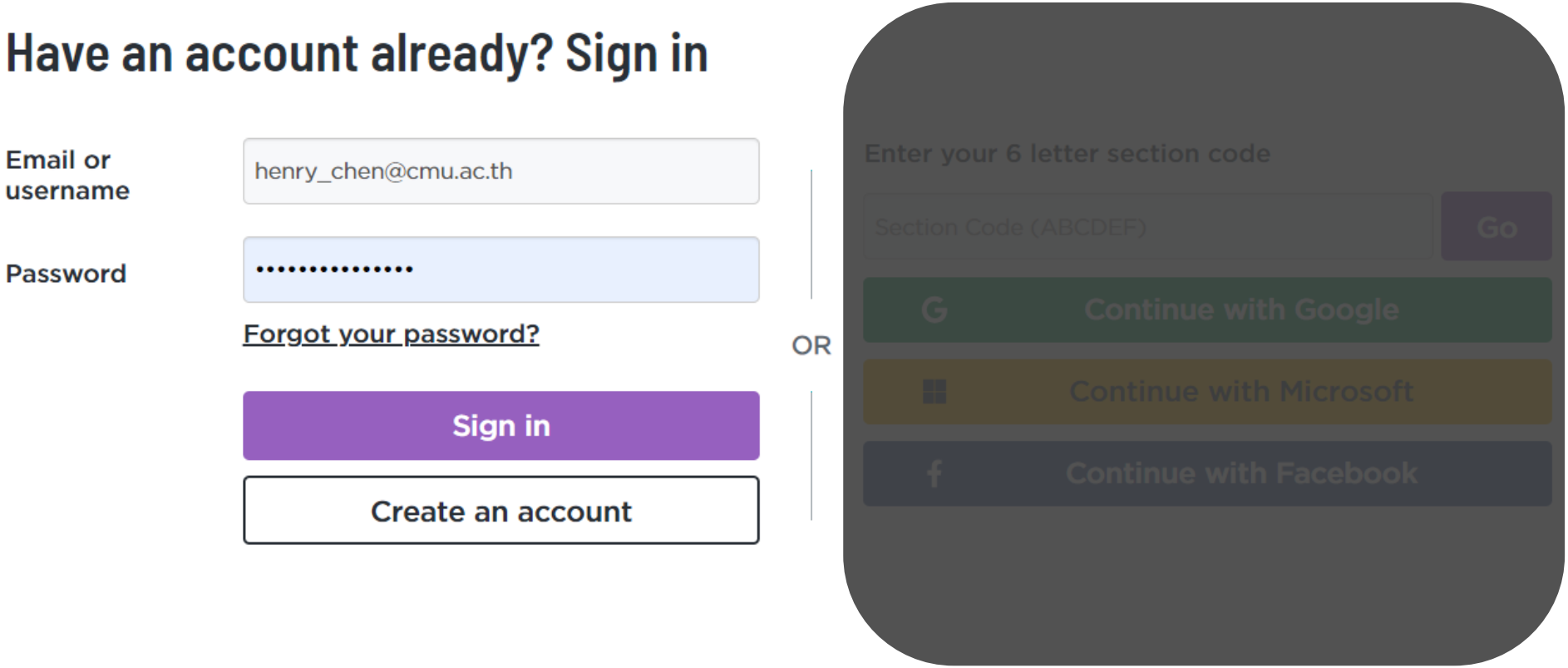
Lesson Name	Progress
1. Maze	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

click to start lesson

Level Type	Level Details			Level Status			
				Not started	In progress	Completed	Assessments / Surveys
Concept	Text	Video	Map	◇	◇	◆	N/A
Activity	Unplugged Lesson Extras	Online Assessment	Question Choice level	○	○	●	●

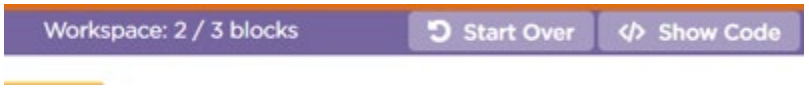
Login in another time…

- Remember username and password to login next time to keep your progress



Lab01

- Finish HOC Classic Maze following level **in class***:
  - Level 1-5 (Sequential Programing)
  - Level 6 (Repeat with the number of times)
  - Level 10 (Repeat with until condition)
  - Level 14 (Selection with condition)
  - Level 18 (Selection with condition and alternative path)
- Study the code with Show Code button



- Challenge yourself by make sure to use blocks as allowed

Homework 1

1. Finish all 20 Levels on HOC
2. Upload Certificate from code.org to Mango assignment

- Deadline*: For all sections
  - 701, 702 and 703: July 15, 2025 (Tuesday)

*Your homework assignment will be graded by midnight on the due date
We don't accept late submissions.

Lab 02

Program Development

Outline

Program Development Cycle

1. Problem Analyses
2. Program Design
3. Coding
4. ~~Debugging~~
5. ~~(Input) Validation~~
6. ~~Documentation~~
7. ~~Program Maintenance~~
8. Input-Output and Variable

- Note that the steps here are not necessarily once-and-done
- You may have to visit earlier steps later one
- Some step (such as documentation), you will have to perform throughout the development cycle

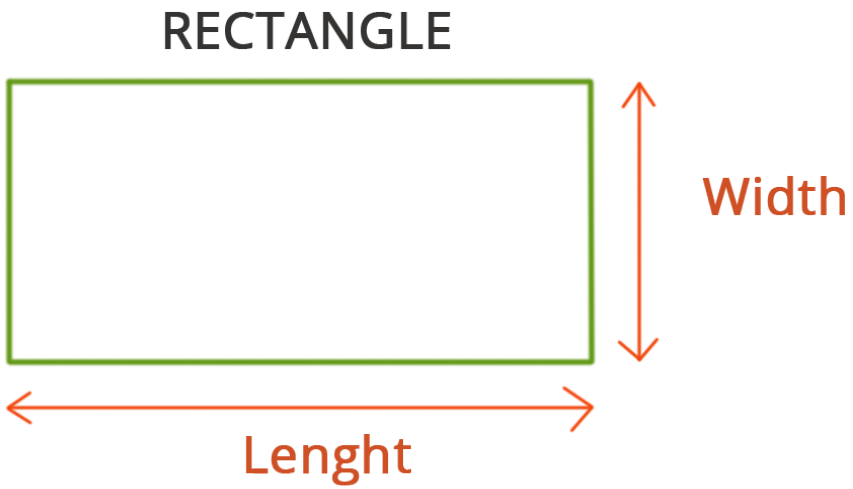
Problem Analyses

- Taking the problem, usually written in human languages, and:
  - Identify components of the problem (input, output, process)
  - Identify processing steps in solving the problem
- Analysis tasks:

Output:	Input:	Process:
The result(s) we want	Data we start with	Steps to transform input data into desired output

1. Problem Analyses – Example #01

- Problem:
  - "Find the area of the specified rectangle, given its length and width."



Area of rectangle = length x width

1. Problem Analyses – Example #01

- Problem:
  - "Find the area of the specified rectangle, given its length and width."
- Analysis Result

Output:	Input:	Process:
The area of the rectangle	**Length and width of the rectangle**	**length * width**
-Which Unit? **cm^2, m^2, $inch^2$, acre?**	**-Which Unit?** **Same as desired output?**	**-Unit conversion?**

2. Program Design

- Lay out the structure of the program
  - Step-by-step working of the program
- Somewhat like architectural plan
  - The building isn't built yet, but you know what it looks like.
- You will use design tools
  - Algorithm Description
  - Pseudocode
  - Flowchart

2. Program Design – Algorithm Description

- Describing program's working in human language
- Describe the process in clear, unambiguous steps
- Example: finding the area of the rectangle

1. Receive the length of the rectangle
2. Receive the width of the rectangle
3. Calculate the area of the rectangle using the following formula:
   - $Area = Length \times Width$
4. Display the resulting area of the rectangle

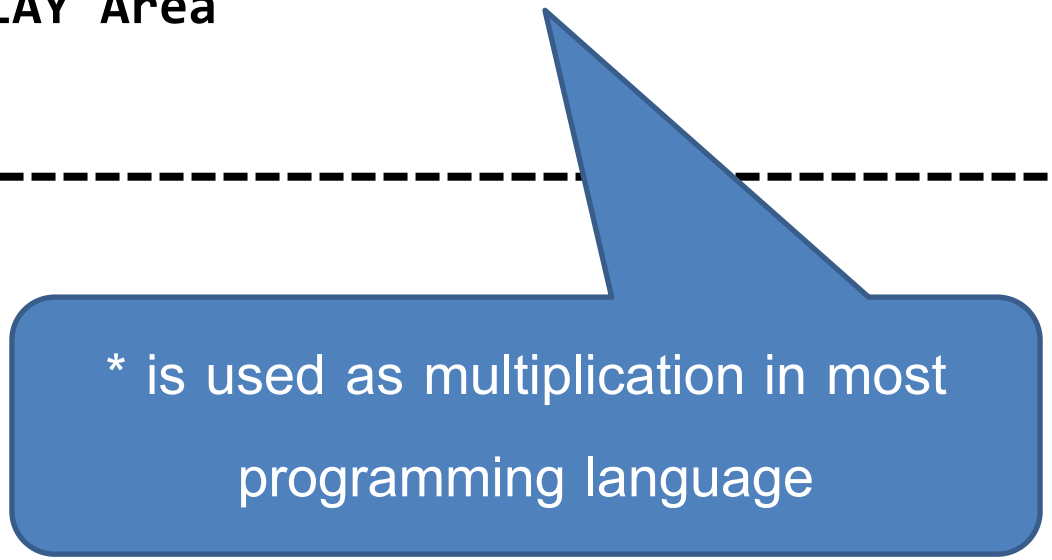
2. Program Design – Pseudocode

- Describe the process in programming-language like description
  - Make sure everyone in the team agree on the pseudocode syntax!
- Specify variables for storing
  - Input data - length (cm) and width (cm)
  - Output result(s) - area (cm^2)
  - (If needed) Intermediate result(s)

2. Program Design – Pseudocode Example

- Example: Finding area of a rectangle

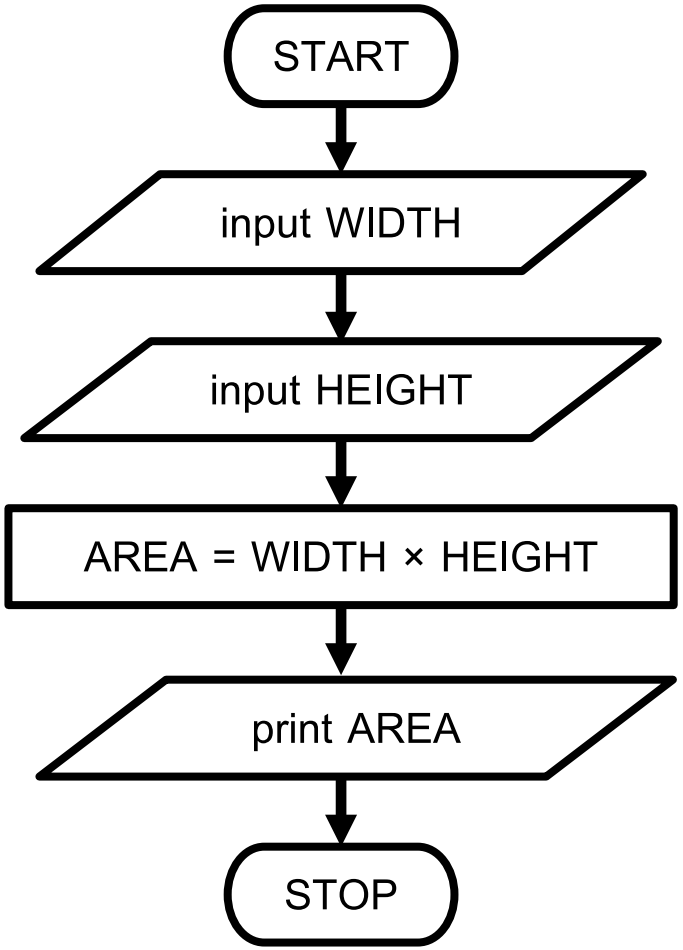
```
START
      READ width
      READ height
      CALCULATE Area = width * height
      DISPLAY Area
STOP
```



2. Program Design – Flowchart

- Describe the process by symbols
- Direction line (→) will show the order of execution among the steps (blocks)
- Easy the see the overall working of the program

Example: Calculating area of a rectangle



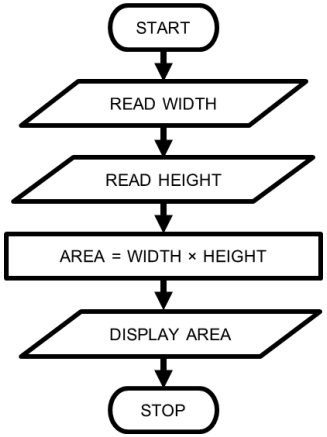
3. Coding

- Writing the program using programming language (such as Python)
- Translate the program design into computer language code
- The code then can be translated into working computer program

3. Coding (cont.)

- Example: translate flowchart for calculating rectangle area into Python code

```
START
        READ width
        READ height
        CALCULATE Area = width * height
        DISPLAY Area
STOP
```




```
width = int(input())
length = int(input())
area = width * length
print(area)
```

Python Interpreter

- The Python interpreter used in this class is CPython (https://www.python.org/), which is the reference implementation created by Guido van Rossum, the creator of the Python language. CPython is the most widely used implementation.
- In addition to CPython, there are other Python interpreters available
  - Jython, written in Java for the JVM,
  - PyPy, written in RPython
  - IronPython, written in C#
- The CPython interpreter has two modes: the Command-line Mode and the Script Mode.

Interactive Mode

- Python offers an interactive mode for interactive coding and immediate feedback.
- Interactive mode is commonly used for testing small code snippets, exploring Python features, and learning the language interactively.
- Online Version of Python Interactive Interpreter
  http://repl.it/languages/Python

```
$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> print(1 + 1)
2
```

Script Mode

```
$ python3 test.py
2
```

- Python offers a script mode for executing Python programs.
- Script mode allows the execution of multiple Python statements or a complete program.
- In script mode, Python programs are typically stored in plain text files with a .py extension.
- To run a Python script, open the terminal or command prompt, navigate to the script's directory, and enter the command "python script_name.py".
- Script mode is commonly used for writing and running standalone Python programs or scripts.

Installing Python

- Version 3.8.10 (Same version to the Grader)
- Integrated Development Environment (IDE)
  - Python provided built-in IDE named IDLE
  - Or you can use another Text Editor
    - Visual Studio Code and PyCharm are recommended

Grader

- We will use an automated grader for programming assignment
- Username: (your Student ID) and passwords are sent to

- Grader Link
  - https://cmu.to/grader204101

Grader Example 01

“Hello World”

Grader Example 01: “Hello World”

- Create and Upload to HW01_0 on the course grader:

  Hello World

```
01 #!/usr/bin/env python3
02 # first_name last_name
03 # 6XXXXXXXX
04 # HW01_0
05 # 204101 Sec 701
06
07 print("Hello World")
```

must have for all submission

Input-output Statements

- For a program to be able to interact with the user, the program need a way to:
  - Receive data from the user (input), and
  - Display result to user (output)
- We will start with:
  - **Output:** `print()` function
  - **Input:** `input()` function

print()

- `print()` will display its arguments (values inside the parentheses) on interactive mode windows.

Code:

```
print ("Welcome ")
print ("to python" )
```

Output:

```
Welcome
to python
```

print()

- `print()` without the automatically added newline character, you need to set the end parameter of the print() function to an empty string (`""`)

Code:

```
print ("Welcome ", end="")
print ("to python" )
```

Output:

```
Welcome to python
```

print()

You can print multiple strings/variables at the same time by separating them with comma (,). The texts will be concatenated when printed.

Code:

```
name = 'ICDI'
ver = 3.9
print ("Hello ! " , name , "Welcome to python" , ver)
```

Output:

```
Hello ! ICDI Welcome to python 3.9
```

Grader Example 02

Compute a times b

Variables

- Variables are named containers used to store data in a program.
- Each variable has a name and a value.
- Variables can hold different types of data, such as numbers, text, or Boolean values.

Variables

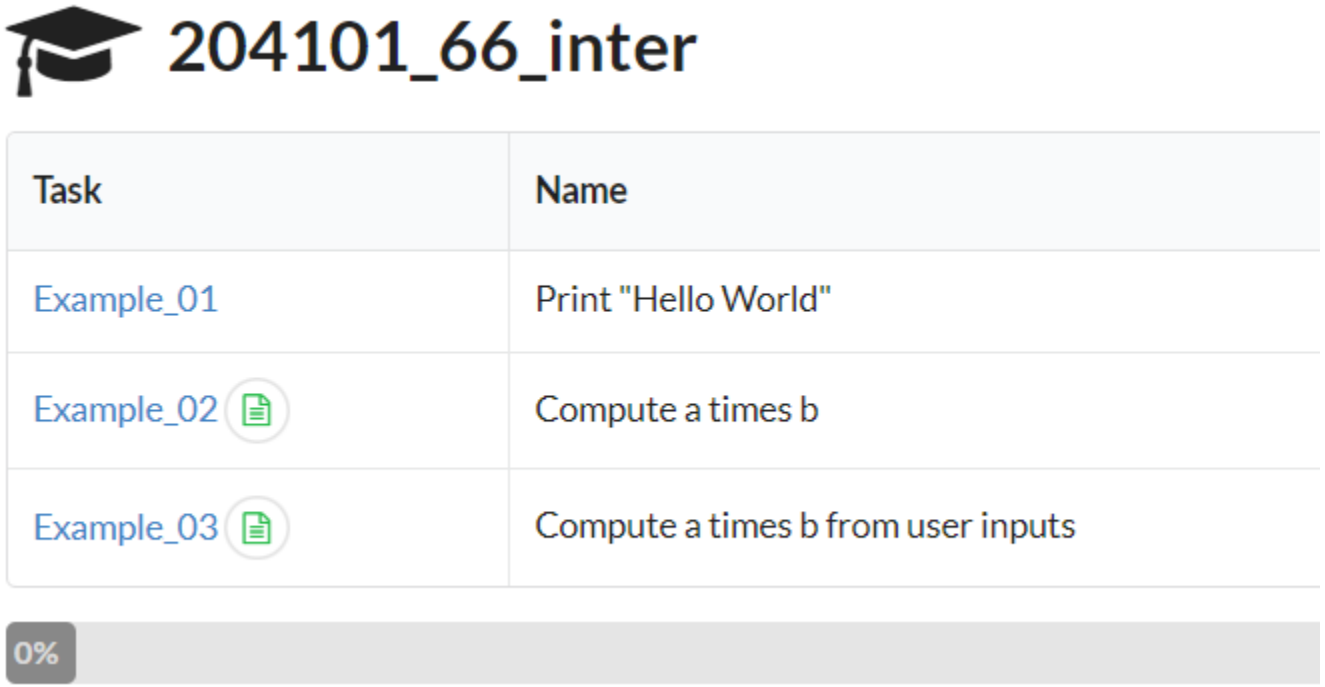
- To create a variable, you use an assignment statement, such as `variable_name = value`

```
05 pi = 3.14
06 radius = 11
07 area = pi * radius * radius
```

- The value of a variable can be changed throughout the program.
- Variables are used to store and manipulate data, perform calculations, and keep track of information.

Grader Example 02

- Compute a times b
- click at instruction on grader and download skeleton file



Compute a times b

Example_02

Task Details

Type	Batch
Time limit	1 second
Memory limit	16 MiB

Submission

Submit your code

Statement

No statement available

Attachments

EX02-skeleton.py

Python script 63 B

1. download the skeleton code

2. write your solution on code and save

3.Upload your code to check for score

Grader Example 03

Compute a times b from user inputs

8.2 input()

- `input()` function will prompt user (with the message inside the parentheses) to enter data. User will type out data, then press enter.
- Received input will be of string type, and we need a variable to hold the data (function return)

Example:

```
str = input("Enter your input: ")
print ("Received input is:", str)
```

Result (user input in blue):

```
Enter your input: Hello Python
Received input is: Hello Python
```

- From the example, "`Hello Python`" will be stored in variable `str`

8.2 input() (cont.)

- Since the input will be string (text),

  it cannot be used in calculation right away

- We will need to convert the input into appropriate type first

  (int or float)

Example:

```
# prompt user for input and store input in variable in_string_1
in_string_1 = input("Input  integer number: ")

# convert text input into integer and store on variable int_val
int_val = int(in_string_1)

# get next user input and store into variable in_string_2
in_string_2 = input("Input  float number: ")

# convert to float, then store it on variable float_val
float_val = float(in_string_2)
```

8.2 input() (cont.)

- Input and convert type (It behaves the same way as the code in the previous slide, but this is a shorter version)

Example:

```
# prompt user for input, convert to int then store in var_1
int_val = int(input("Input  integer number: "))

# get another input and convert to float, then store in var_2
float_val = float(input("Input  float number: "))
```

8.2 input() (cont.)

- Warning: be careful about input data type.

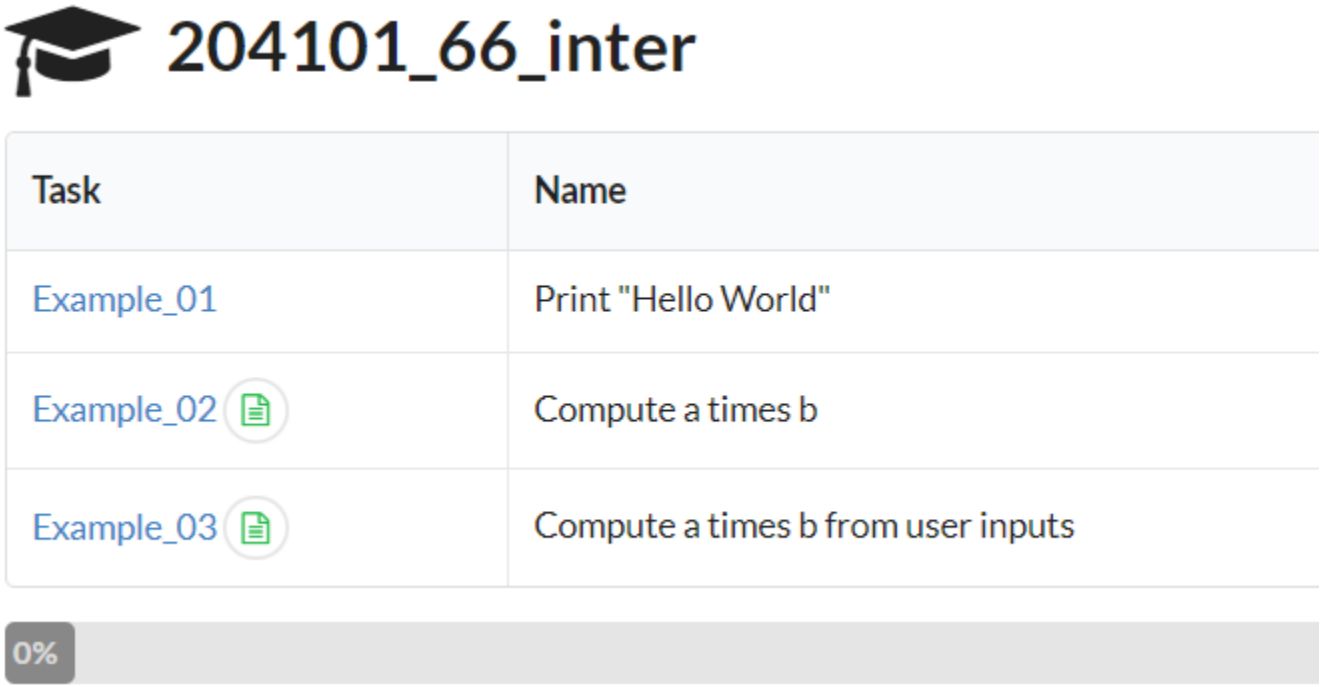
Example:

```
>>> inp = int(input("input an  integer: "))
input an  integer: 1.2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '1.2'
>>>
```

- If the user input 1.2, it will read and put into inp with problem. BUT when Python try to convert it to integer, an error will occur (**since 1.2 is not an integer**)

Grader Example 03

- Compute a times b from user inputsclick at instruction on grader and download skeleton file



Lab Assignment

- Complete 3 tasks on course grader as followed

204101_66_inter

Task	Name	Status	Public Score	Time limit	Memory limit
Example_01	Print "Hello World"	Evaluated	1 / 1	1 second	16 MiB
Example_02	Compute a times b	Evaluated	1 / 1	1 second	16 MiB
Example_03	Compute a times b from user inputs	Evaluated	5 / 5	1 second	16 MiB

100%

Lab Exercise

One Last Exercise For Lab 02 (Not graded)

(**Lab Exercise**) Write a program that prompts the user to enter their name, surname, and age. The program should then output the provided information, along with the age at which the user will graduate (assuming a standard four-year duration). **If you are done, show your code to the instructor and/or the TA for review.**

Input (user input in red)

```
Input name: Chalee
Input surname: Buddee
Input age: 18
```

Output

```
My name is Chalee
And my surname is Buddee
Now I'm 18 years old
I'll graduate when I'm 22
```

CS204101

How to use the Grader Platform

Grader = An Automated Grading System

Faculty of Science, Chiang Mai University
คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่

Get your Grader Password

- Scan QR code via your phone QR code scanner (NOT from CMU Mobile APP)

Get you Grader Password

- Your Grader username is your student ID
- Your Grader password is here

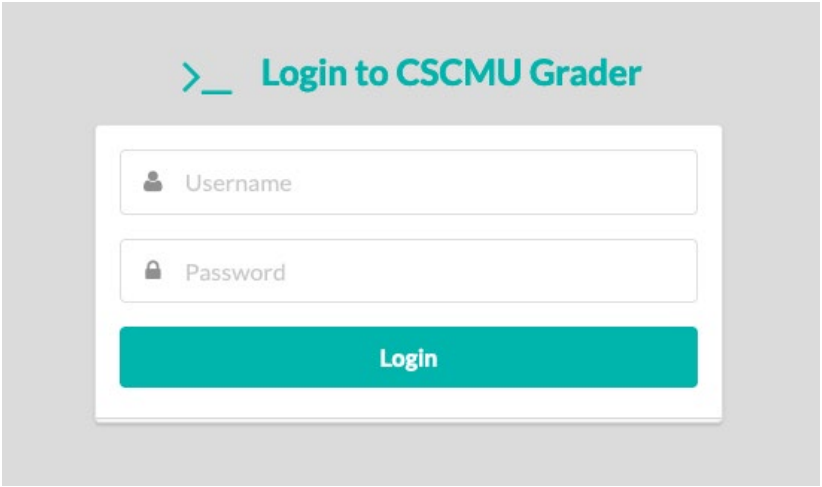
Access The Grader Platform

1. Open your preferred web browser (Google Chrome, Mozilla Firefox, Safari, etc.)

2. Type the following URL in the address bar and hit enter.

https://grader101inter.cs.science.cmu.ac.th/

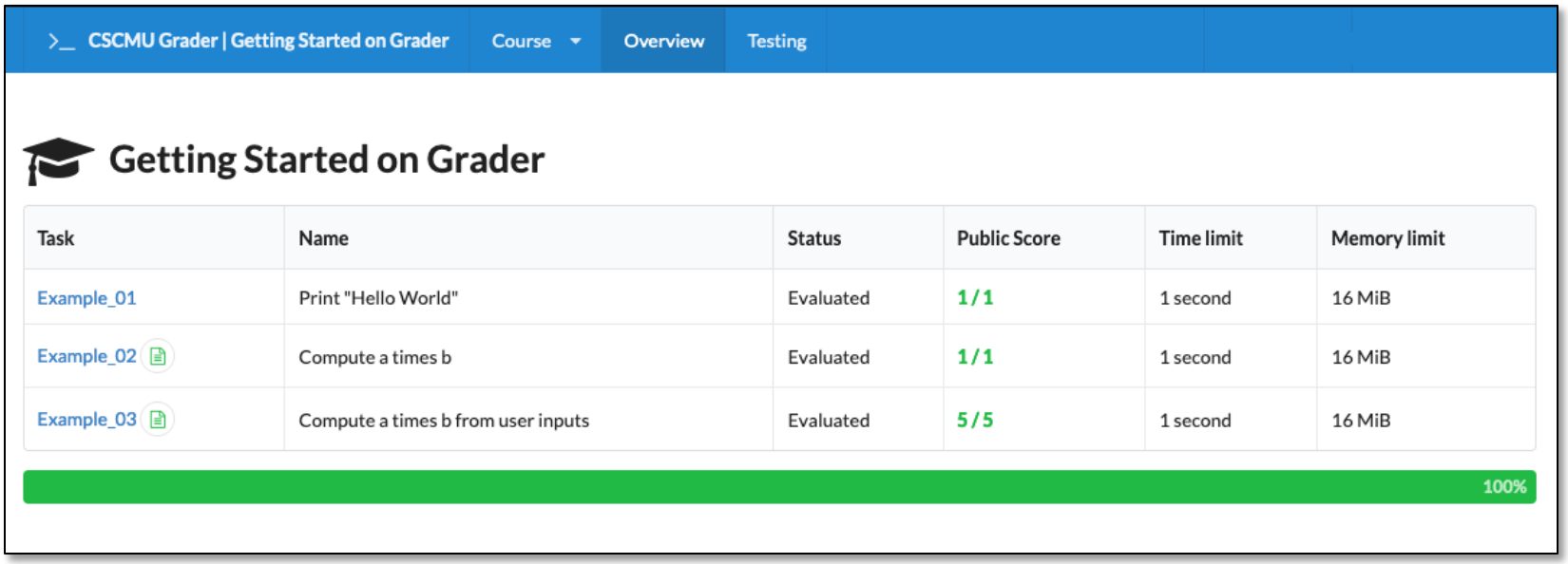
3. You should see the login page



Login To Your Account

1. On the login page, enter your username and password given to you (if you don't receive or can't find one, contact your instructor)

2. Click the login button to access your account. You will then be redirected to the landing page upon successfully logging in



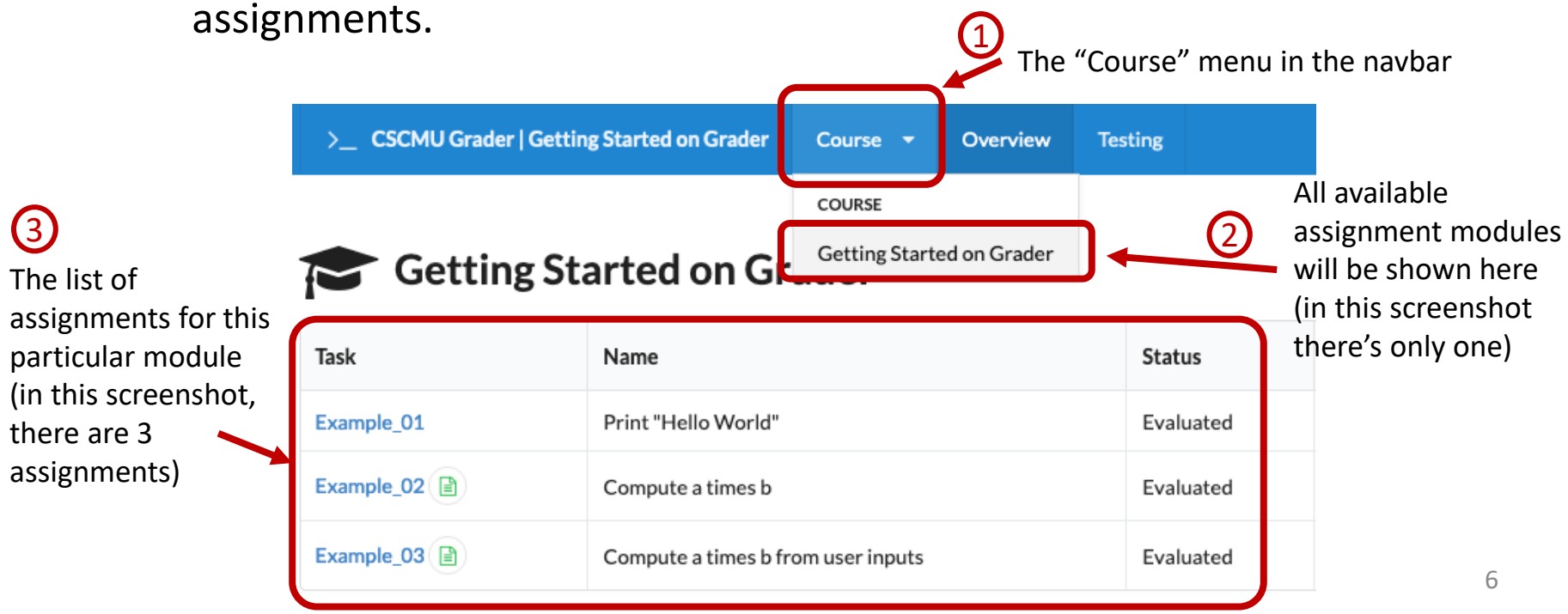
The landing page
(yours may look different from this)

Select An Assignment Module

1. Look for the “Course” menu in the navigation bar at the top of the page.

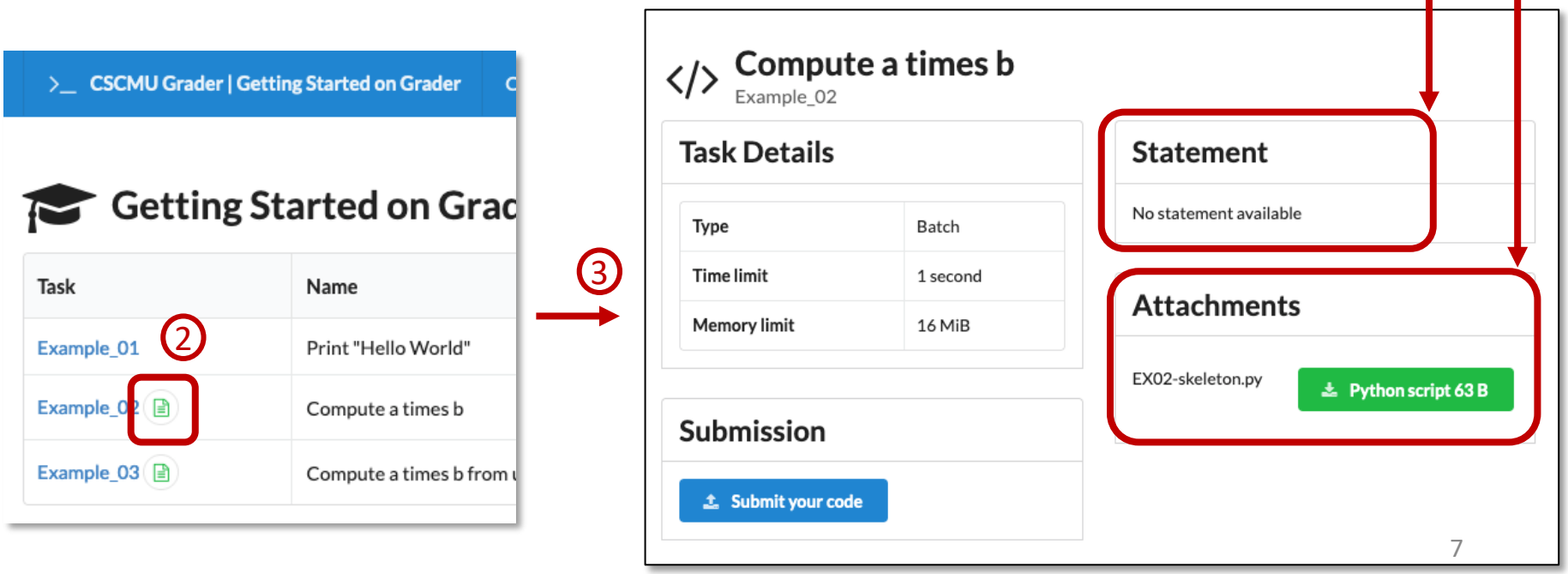
2. Hover your mouse over or click on the menu to view all available assignment modules.

3. Select the assignment module for which you want to see the list of assignments.



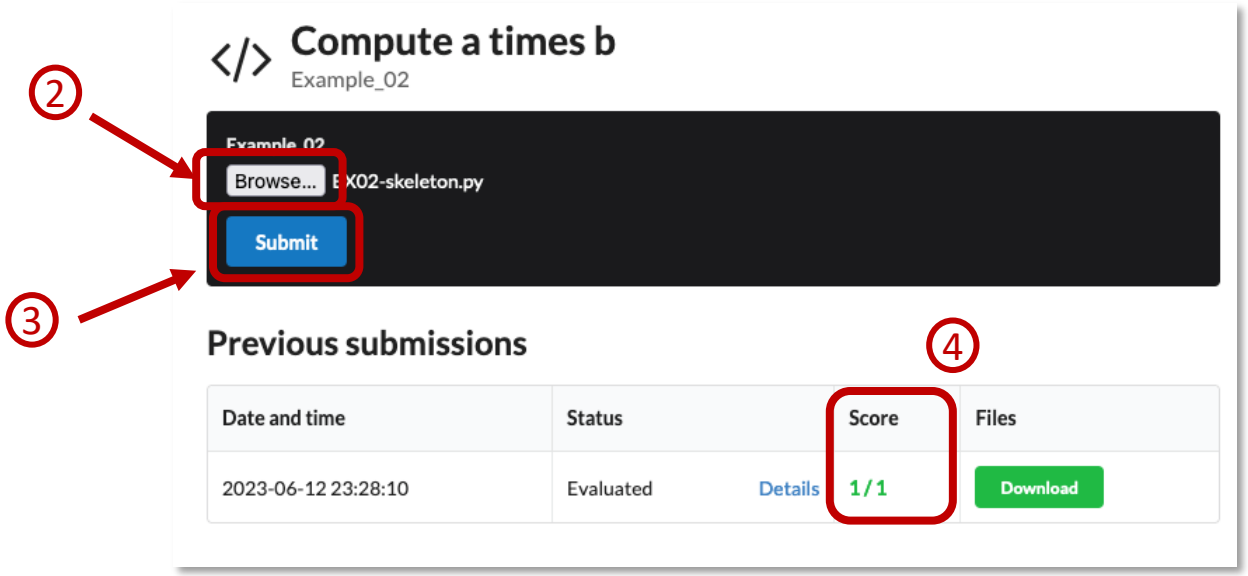
Download Instructions and Starter Code

1. Most assignments will come with instructions (PDF) and/or starter code (.py).
2. Look for a green icon next to the assignment ID (in the “Task” column) and click on it.
3. The assignment’s instructions will be located in the “Statement” section, while the starter code will be located in the “Attachments” section.



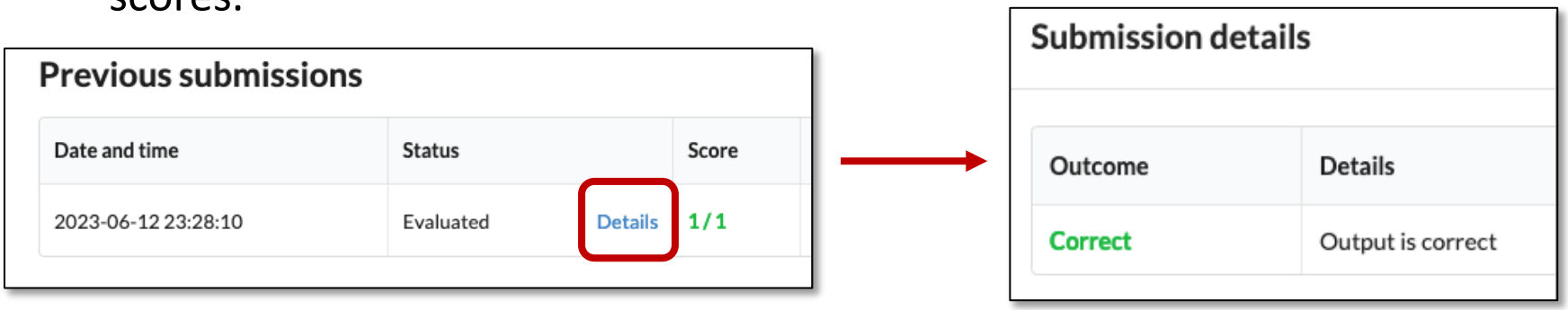
Submit An Assignment

1. To access the submission page, you can click the assignment ID on the list of assignments page.
2. Click “Browse” to select the file from your computer for submission.
3. Click the “Submit” button.
4. Once the grader has finished running, you should be able to see your grade for that assignment. (The grader may take some time to run your code.)



Basic Troubleshooting

- You can click on “Details” to view more information on your submission scores.



- You will see one of two outcomes: **Correct** or **Not Correct**.
- In the case of “**Not Correct**”, the “Details” column will provide one of two reasons:

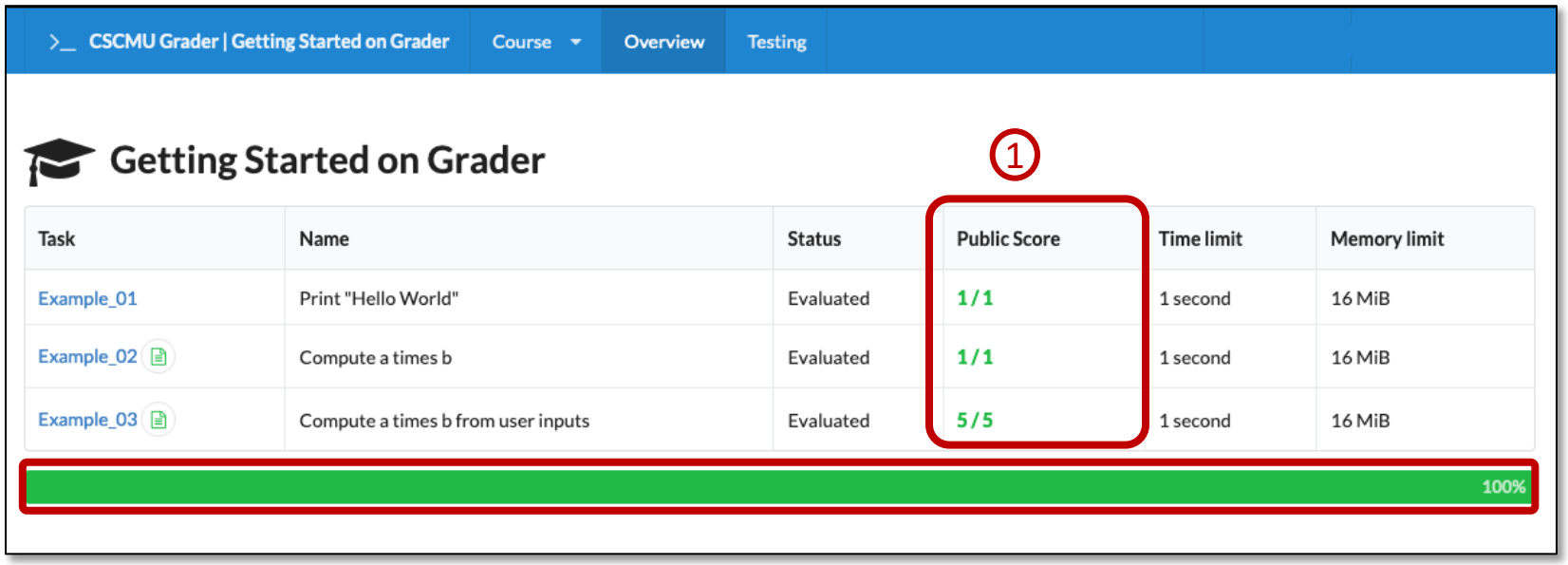
1. **Output isn’t correct**. This means that the Grader was able to run your code, but the code didn’t produce the expected output.

2. **Execution failed because the return code was nonzero**. This means that your code cannot be executed due to an error.

Review Grades

1. If you go to the page with all the assignments for the module you selected, you'll see a list of assignments you have completed, along with the scores for each of them.

2. The progress bar at the bottom of the table provides a visual aid to help you track your scores for the module.



W03 Lab

Program Development (cont.)

Exercise

Write a program that prompts the user to enter their name, surname, and age. The program should then output the provided information, along with the age at which the user will graduate (assuming a standard four-year duration).

Input (user input in red)

```
Input name: Chalee
Input surname: Buddee
Input age: 18
```

Output

```
My name is Chalee
And my surname is Buddee
Now I'm 18 years old
I'll graduate when I'm 22
```

Defining Input - Process - Output

Input (user input in red)

Input name: Chalee
Input surname: Buddee
Input age: 18

Output

My name is Chalee
And my surname is Buddee
Now I'm 18 years old
I'll graduate when I'm 22

Input	Process	Output

- Retrieve data from keyboard
- Create variable to store data
  - name
  - lastname
  - age
	- Calculate graduated_age by
	  - age + 4
		Display
		- Specific dialog with value in variables
		  - name
		  - lastname
		  - age
		  - graduated_age

Create an algorithm flow

1. Prompt user for name
2. Read name
3. Prompt user for surname
4. Read surname
5. Prompt user for age
6. Read age

Input

7. Calculate graduation age (age + 4)

Process

8. Print name, surname, age, and graduation age

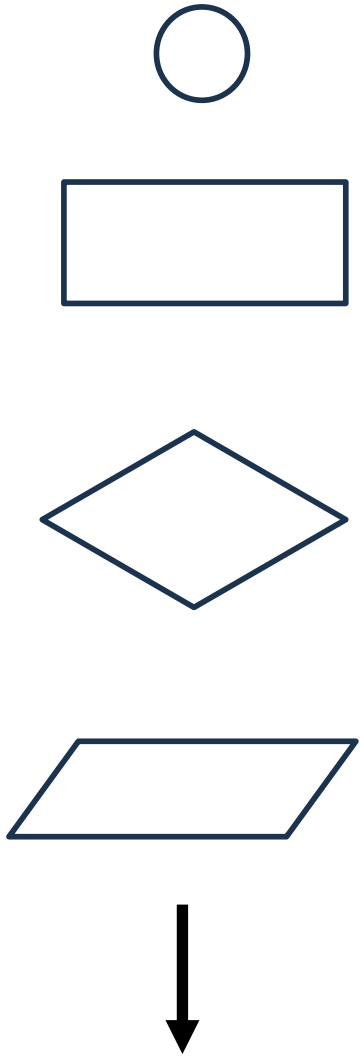
Output

Flowchart

- A flowchart is a graphical representation of a process, algorithm, or system.
- It uses different shapes and arrows to depict the sequence of steps or actions in a logical and visual manner.
- Flowcharts are widely used in various fields, including computer programming, business process management, system analysis, and problem-solving.

Flowchart Symbols

1. Start/End: Indicates the beginning or end of the process.
2. Process/Action: Represents a specific action or process.
3. Decision: Represents a decision point where the flow can branch into different paths based on a condition.
4. Input/Output: Represents input or output of data or information.
5. Connector: Connects different parts of the flowchart.



Algorithm Flow with Flowchart

1. Prompt user for name
2. Read name
3. Prompt user for surname
4. Read surname
5. Prompt user for age
6. Read age

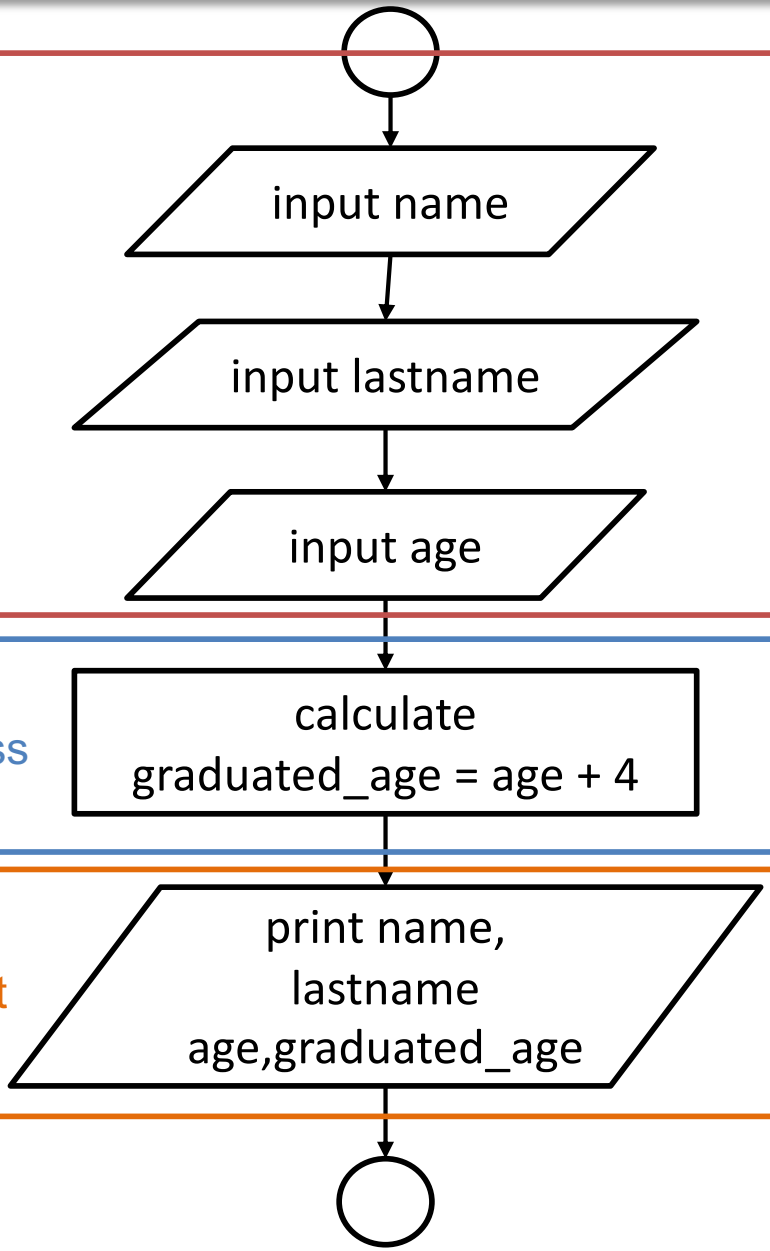
Input

7. Calculate graduation age (age + 4)

Process

8. Print name, surname, age, and graduation age

Output



Transform Algorithm to Code

```
01 # Prompt the user to enter their name
02 ______ = input("Input name: ")
03
04 # Prompt the user to enter their surname
05 ______ = input("Input surname: ")
06
07 # Prompt the user to enter their age
08 age = _______(input("Input age: "))
09
10
11 # Calculate the graduation age assuming a four-year duration
12 graduation_age = age ___________
13
14 # Print the user's information and projected graduation age
15 print("My name is ", name)
16 print("And my surname is ", surname)
17 print("Now I'm ", age, "years old" )
18 print("I'll graduate when I'm ", _________________)
19
```

Input

Process

Output

Practice 1: Fahrenheit to Celcius

Practice 1: Fahrenheit to Celsius

- To convert a temperature value from Fahrenheit to Celsius, you can use the following formula:

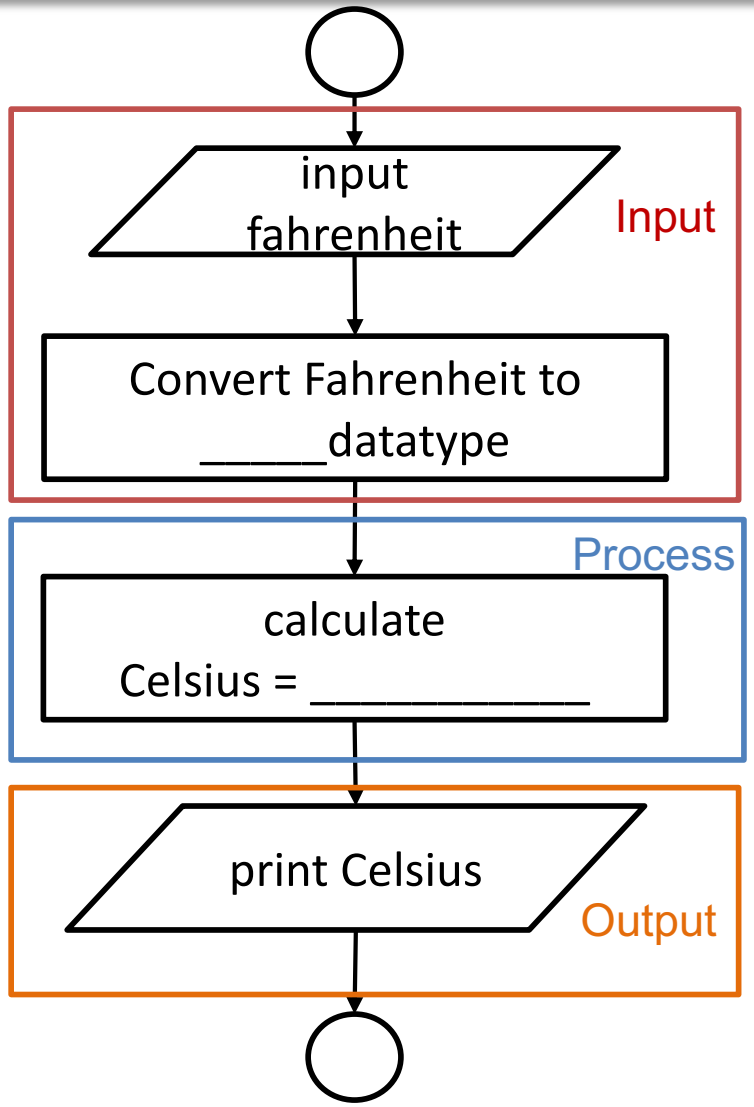
$$\frac{C}{5} = \frac{F - 32}{9}$$

- Where:
  - C is the temperature in Celsius
  - F is the temperature in Fahrenheit
- **Input:**
  - Take in input F using `input("Input temperature in Fahrenheit: ")`
- **Process:**
  - convert F to float by using `float(F)`
  - calculate C using the formular
- **Output:**
  - print out the result as followed

```
Input temperature in Fahrenheit: 50
The temperature in Celsius is: 10.0000
```

Algorithm Flow

- input has string datatype
- to use in calculation, conversion is needed
- from the formula $\frac{C}{5} = \frac{F - 32}{9}$
  - C = _____



Transform Algorithm to Code

```
01 # Prompt the user for temperature in Fahrenheit
02 fahrenheit = input("Enter the temperature in Fahrenheit: ")
03 fahrenheit = ______(fahrenheit)
04 # Convert Fahrenheit to Celsius using the conversion formula
05 celsius = ____________________
06
07 # Print the Celsius temperature
08 print("The temperature in Celsius is:", ____________)
09
10
11
```

Input

Process

Output

$$\frac{C}{5} = \frac{F - 32}{9}$$

If you're not sure where to start, check out the flow diagram from the previous slide.

When you are done, show your code to the instructor or the TA for review

Practice 2: BMI Calculation

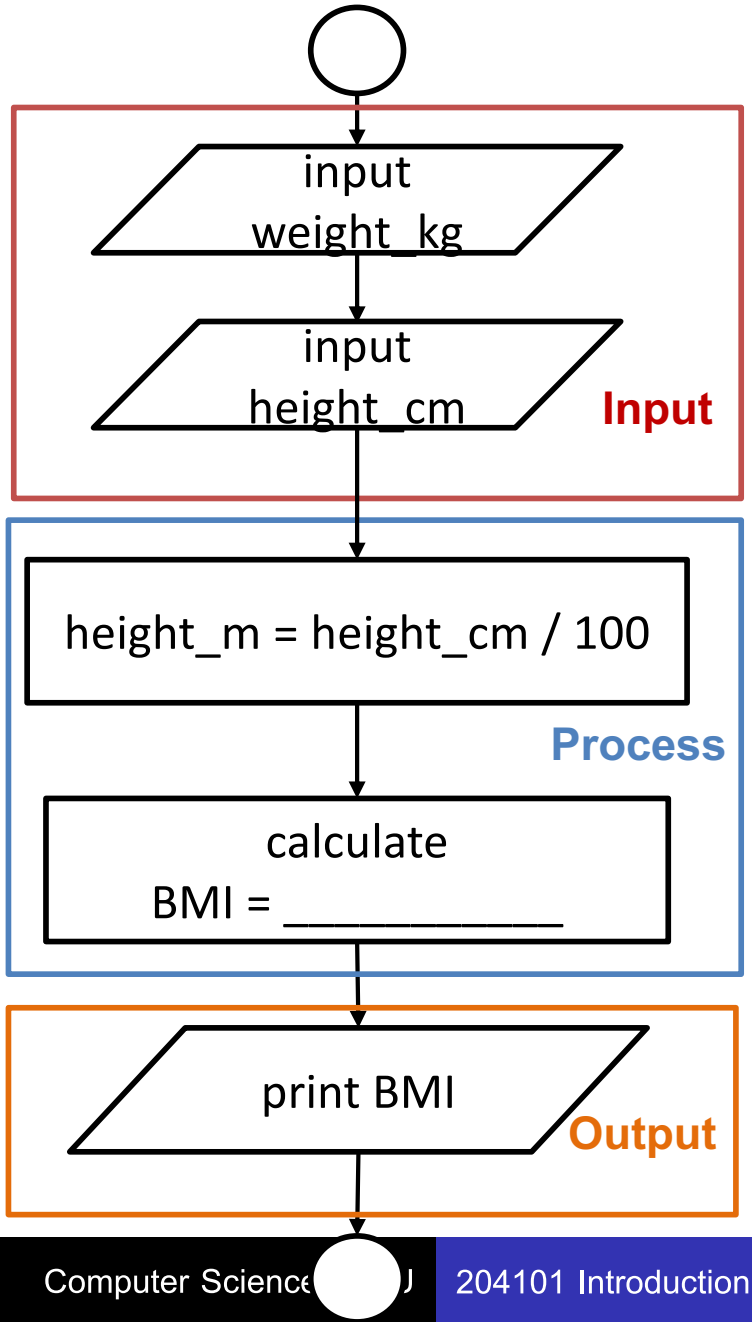
Practice 2: BMI Calculation

- Write a program that asks the user to enter their weight in **kilograms** and their height in **centimeters**.
- The program should then calculate and print their BMI (Body Mass Index).
- The BMI calculation is based on the formula: BMI = weight / (height^2), where
  - weight is measured in kilograms
  - height is measured in meters.
  - 1 meter = 100 centimeters

Algorithm Flow

1. Prompt the user to enter their weight in kilograms.
2. Read the user's _____.
3. Prompt the user to enter their height in centimeters.
4. Read the user's _____.
5. Convert height from **centimeters** to **meters** by using formula height_m = _____
6. Calculate the BMI using the formula
   BMI = _____
7. Print the calculated BMI.

Algorithm Flow in Flowchart



Transform Algorithm to Code

```
01  # Prompt the user for weight in kilograms
02  weight_kg = _____(input("Enter your weight in kilograms: "))
03
04  # Prompt the user for height in centimeters
05  height_cm = ______(input("Enter your height in centimeters: "))
06
07  # Convert height from centimeters to meters
08  height_m = ____________________
09
10  # Calculate BMI using the formula BMI = weight / (height^2)
11  bmi = ____________________________
12
13  # Print the calculated BMI
14  print("Your BMI is:", bmi)
15
```

Input

Process

Output

When you are done, show your code to the instructor or the TA for review

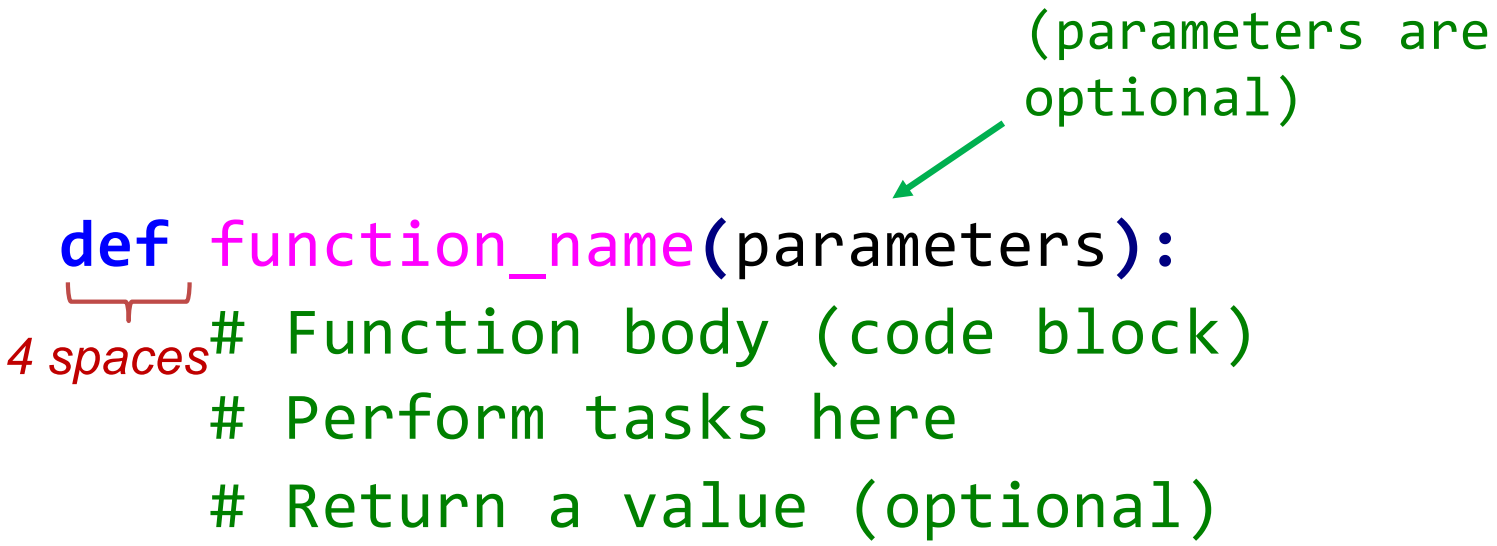
Input: 90
186

Expected: 26.014568158168572

INTRODUCING FUNCTIONS

functions

- In Python, you can define your own functions using the `def` keyword.



Functions

- A function is a section of reusable code that performs a specific task or set of tasks.
- To call a function, type its name followed by a pair of parentheses.
- You can pass data, known as arguments, to the function, but this requires a matching set of parameters in the function's definition.
- A function can return data back to the location where it was called.

Function Example

We define the square function here

```
01 def square(number):
02     result = number ** 2
03     return result
04
05 # Call the square function with an argument
06 num = int(input("Input number: "))   #5
07 result = square(num)
08 print(result)  # Output: 25
```

- Line 01 - 03
  - The function **square** takes a parameter called ***number***.
  - Inside the function, it calculates the square of the number by multiplying it by itself ***(number ** 2)***.
  - Then, it returns the result using the return statement.

Function Example

The square function

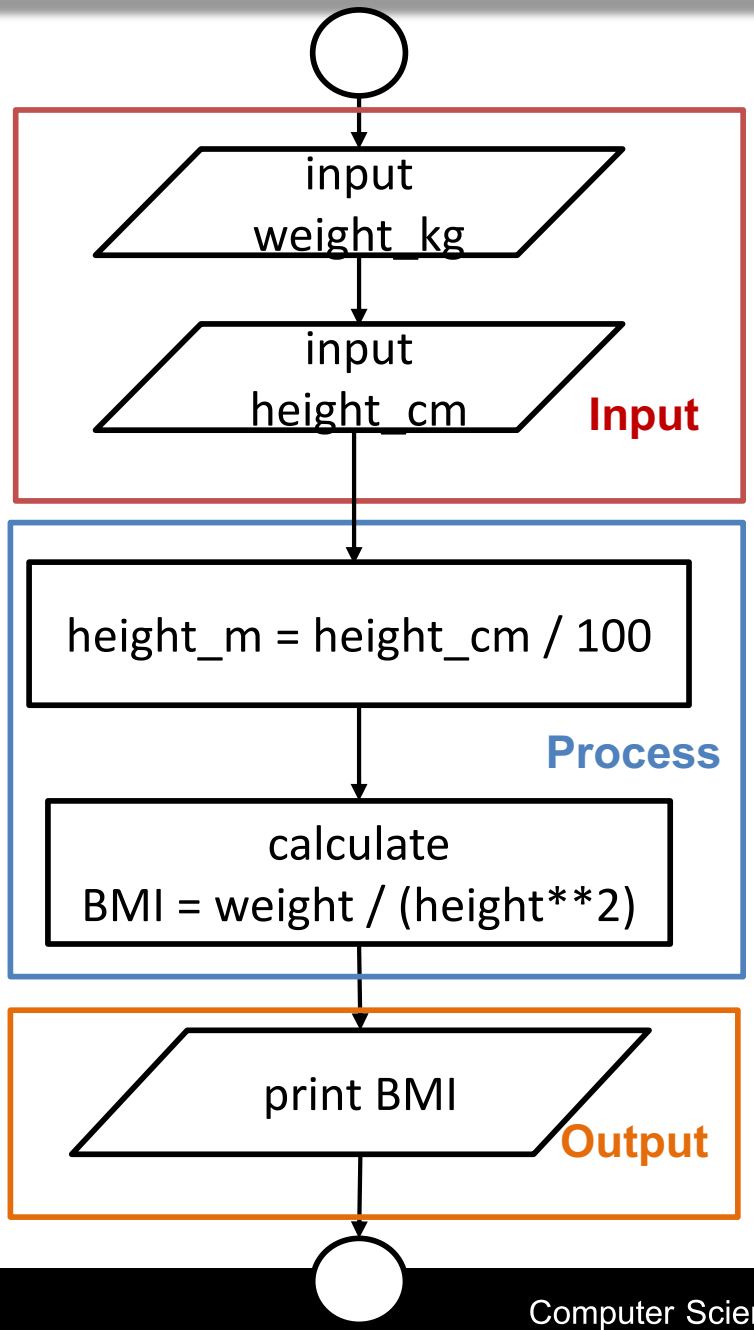
```
01  def square(number):
02      result = number ** 2
03      return result
04
05  # Call the square function with an argument
06  num = int(input("Input number: "))   #5
07  result = square(num)
08  print(result)  # Output: 25
```

Calling the square function to use its process (pass in the value from the user input as an argument)

- Line 06-08
  - we call the **square** function and pass the value 5 as an argument (from the user input).
  - The function executes its code block, calculates the square of 5, and returns the result.
  - The returned result is stored in the ***result*** variable, which is then printed to the console

Let’s modify our code for BMI calculation to include a function.

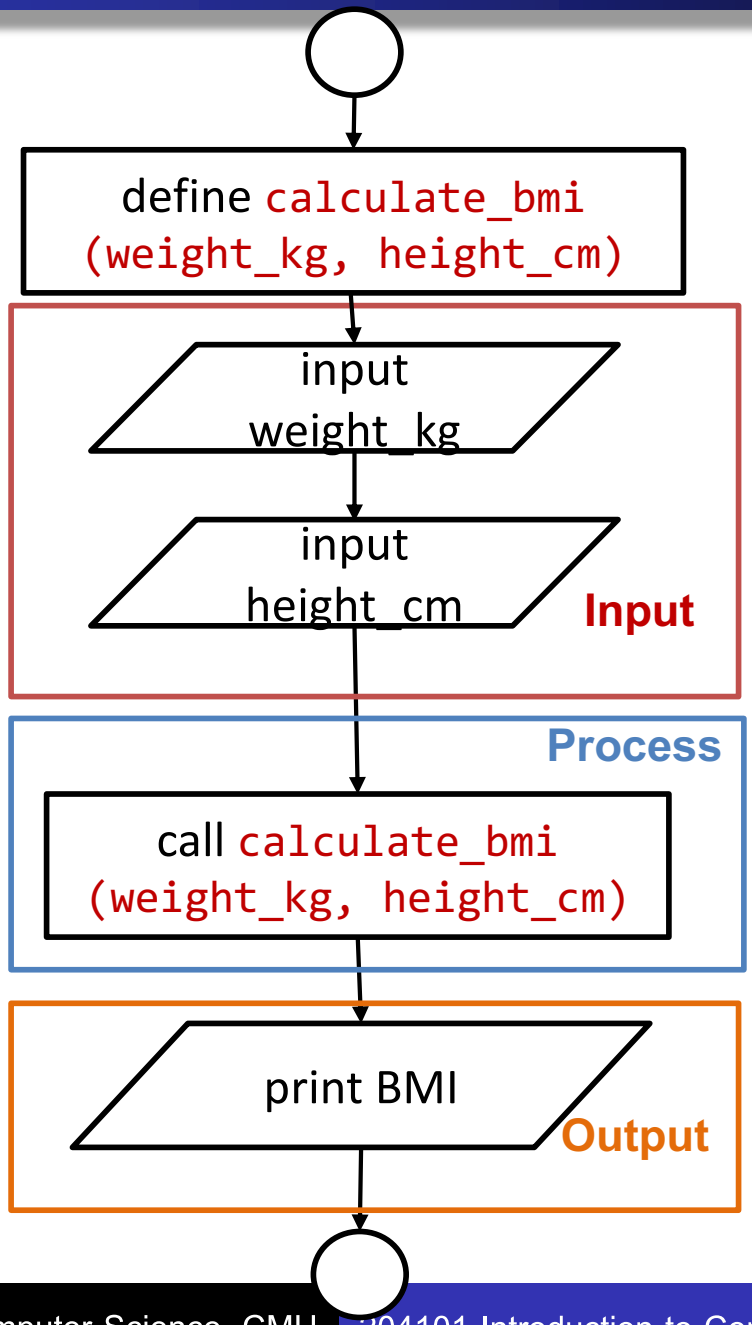
BMI Calculation as a Function



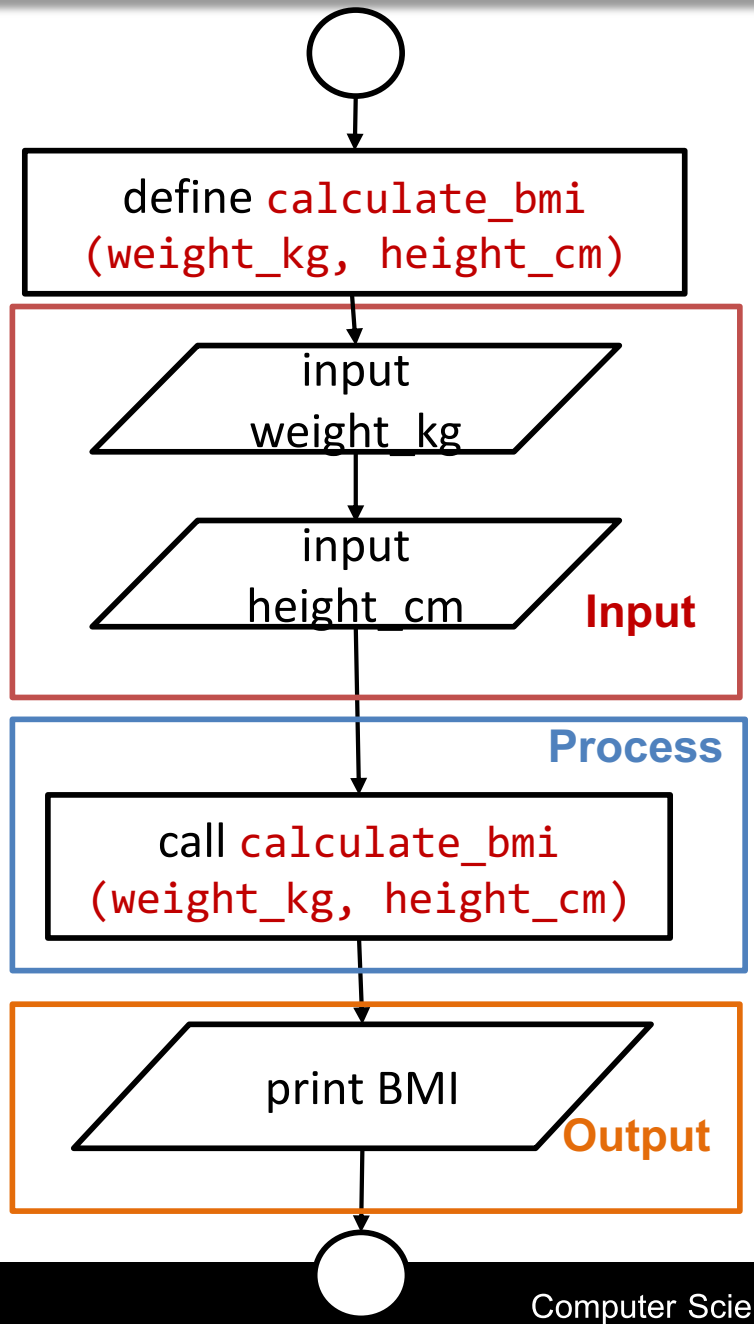
To improve code organization and reusability, we can introduce a function for the BMI calculation.

1. Define a function named `calculate_bmi` that takes two parameters: `weight_kg` and `height_cm`.
2. Inside the function, convert the height from centimeters to meters by dividing it by 100.
3. Calculate the BMI using the formula BMI = weight / (height**2).
4. Return the calculated BMI from the function.

BMI Calculation as a Function



BMI Calculation as a Function




```
01  def calculate_bmi(weight, height_cm):
02      height_m = height_cm / 100
03      bmi = weight / (height_m ** 2)
04      return bmi
```

```
14  bmi = calculate_bmi(weight, height_cm)
```

BMI Calculation as a Function

```
01  def calculate_bmi(weight, height_cm):
02      height_m = height_cm / 100
03      bmi = weight / (height_m ** 2)
04      return bmi
05
06  # Prompt the user to enter their weight in kilograms
07  weight = float(input("Enter your weight in kilograms: "))
08
09  # Prompt the user to enter their height in centimeters
10  height_cm = float(input("Enter your height in centimeters: "))
11
12  # Call the calculate_bmi function with the provided weight and
13  height inputs
14  bmi = calculate_bmi(weight, height_cm)
15
16  # Print the calculated BMI
17  print("Your BMI is:", bmi)
18
```

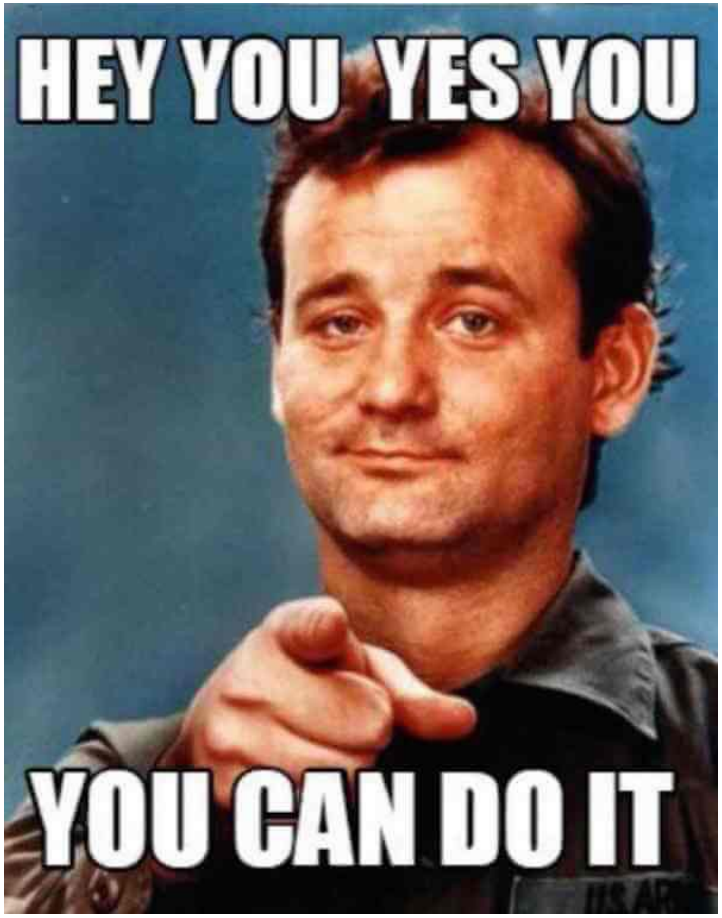
Process:
Function Definition

Input

Process:
Function Call

Output

Now that you've been exposed to the concept of functions, you should be able to complete the lab assignments with ease!



W03 LEC

Defining a Function

Introduction to User-Defined Functions in Python

- Welcome to the lecture on user-defined functions in Python.
- In this lecture, we will learn about the concept of functions and how to create our own functions in Python.
- Functions play a crucial role in structuring our code, promoting reusability, and making our programs more modular.

What are Functions?

- Functions are a way to group code into reusable blocks that perform specific tasks.
- They allow us to break down complex programs into smaller, more manageable parts.
- Functions can take input values called arguments or parameters, perform operations, and return output values.
- Functions are a key way to define interfaces so programmers can share their code.

Why Use Functions?

- Functions provide several benefits:
  - Code Reusability: Once a function is defined, it can be used multiple times throughout the program, eliminating code duplication.
  - Modularity: Functions help organize code into logical blocks, making it easier to understand and maintain.
  - Abstraction: Functions allow us to encapsulate complex operations behind a simple interface, making the code more readable.

Why Use Functions?

- Greeting

```
print("Hello, John")
print("Welcome to our 204101 class")
print("Hope you have fun with Python")

print("Hello, Ann")
print("Welcome to our 204101 class")
print("Hope you have fun with Python")

print("Hello, Dan")
print("Welcome to our 204101 class")
print("Hope you have fun with Python")
```

```
Hello, John
Welcome to our 204101 class
Hope you have fun with Python

Hello, Ann
Welcome to our 204101 class
Hope you have fun with Python

Hello, Dan
Welcome to our 204101 class
Hope you have fun with Python
```

Me, after greeting all 150 students in 204101 class

How do we write functions in Python?

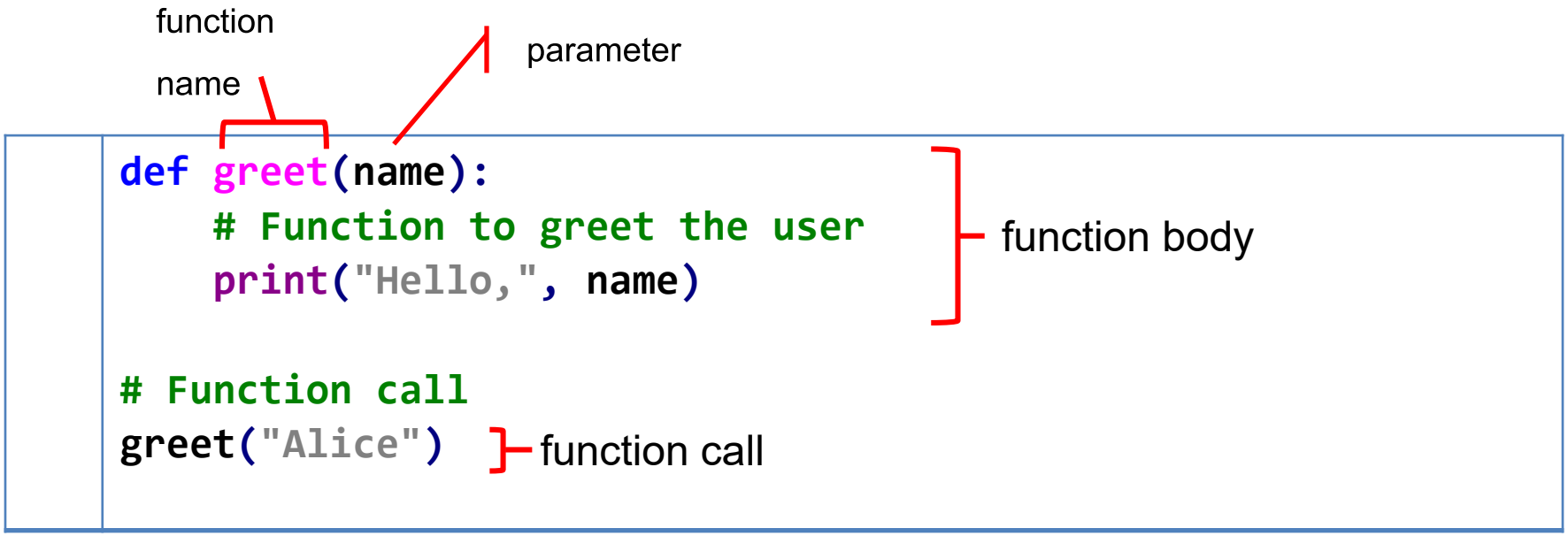
Function Syntax

```
def function_name(param1, param2, …):
    line of code 1
    line of code 2
    …
    line of code N
    return result #Optional
```

4 spaces (or a tab)

- **def**: Keyword used to define a function.
- **function_name**: Name of the function, following Python naming conventions.
- **param1, param2,** ...: Input parameters that the function expects. A function can have zero or many parameters (optional)
- **return**: Keyword used to specify the value the function should return (optional).

Function Example



```
greet("Alice")
```

```
def greet( "Alice"  ):
    # Function to greet the user
    print("Hello,", name     )
```

Output:

Hello, Alice

Why Use Functions?

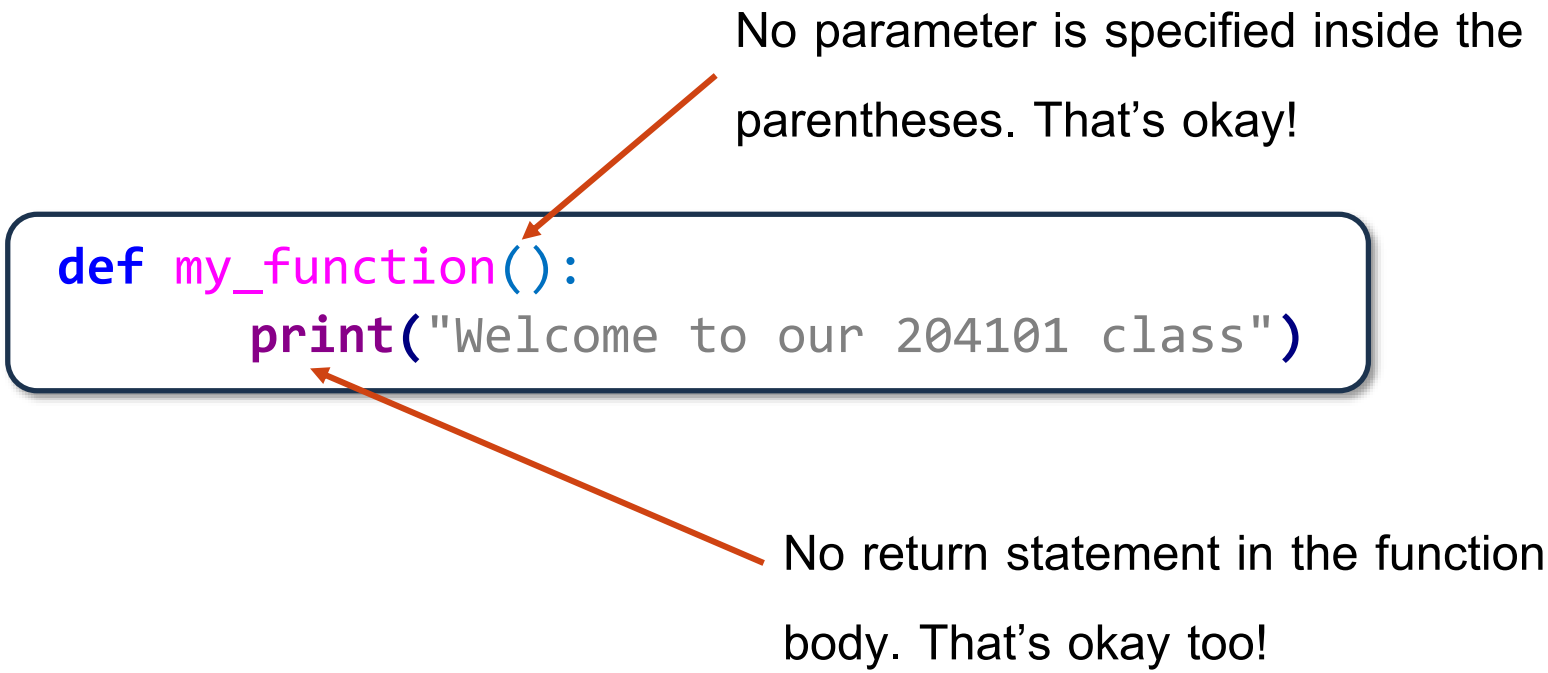
- Greeting

```
def greet(name):
    print("Hello,", name)
    print("Welcome to our 204101 class")
    print("Hope you have fun with Python")

greet("John")
greet("Ann")
greet("Dan")
```

```
Hello, John
Welcome to our 204101 class
Hope you have fun with Python

Hello, Ann
Welcome to our 204101 class
Hope you have fun with Python

Hello, Dan
Welcome to our 204101 class
Hope you have fun with Python
```

Me, after greeting all 150 students in 204101 class using function

- An example of a function with no parameter (and no return statement):



Return Statement

- Functions can return values using the `return` statement.
- The `return` statement specifies the value or expression that the function should return.
- Once a `return` statement is encountered, the function execution terminates and the result is returned to the caller.

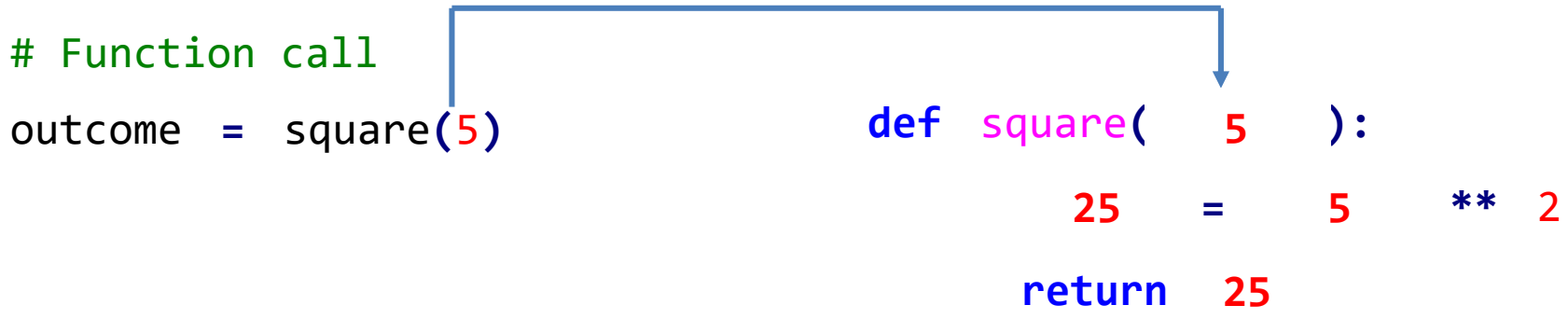
Return Function

```
def  square(number):
      # Function to calculate the square of a number
      result  =  number  **  2
      return  result

# Function call
outcome =  square(5)
print("Square of 5 is",  outcome)
```

- In this example, the **square** function takes one parameter, **number**.
- Inside the function body, we calculate the square of the number and assign it to the **result** variable.
- We then use the **return** statement to return the calculated result.
- The returned value is stored in the **outcome** variable outside the function and displayed on the screen.

Return Function

```
def  square(number):
      # Function to calculate the square of a number
      result  =  number  **  2
      return  result

# Function call
outcome =  square(5)
print("Square of 5 is",  outcome)
```



Check Your Understanding

What is the output of the following code?

```
1  def greet(name):
2      print("Hello, " + name)
3
4  greet("Alice")
5  greet()
```

Check Your Understanding

What is the output of the following code?

```
1  def numval(num):
2      num += 1
3      print("num is: ",num)
4
5  value = 10
6  numval(value)
```

Variable Scope

Variable Scope and Parameter Passing in Python

- In this section, we will discuss the concept of variable scope and how parameters are passed to functions in Python.
- Understanding variable scope and parameter passing is crucial for writing efficient and bug-free code.

Variable Scope

- Variable scope refers to the visibility or accessibility of a variable within a program.
- In Python, variables can have either global scope or local scope.
  - Global variables are accessible throughout the entire program,
  - Local variables are only accessible within a specific block of code, such as a function.

Global Variable

- Global variables are defined outside of any function and are accessible throughout the program.
- They can be accessed and modified by any part of the code.

Global variable

```
# Global variable
global_var = "Hello, I'm a global variable"

def print_global_var():
    # Accessing the global variable inside a function
    print(global_var)

def modify_global_var():
    # Modifying the global variable inside a function
    global global_var
    global_var = "Modified global variable"

# Calling the functions
print_global_var()
modify_global_var()
print_global_var()
```

global keyword to indicate that we want to modify the global variable

```
Hello, I'm a global variable
Modified global variable
```

Global variables should be used with caution to avoid naming conflicts.

Local Variable

- Local variables are defined inside a function and are accessible only within that function.
- They are created when the function is called and destroyed when the function finishes execution.
- Local variables have a limited lifespan and do not interfere with variables outside the function.

Local Variable

- `result` variable is a local variable because **it is defined within the function.**
- It is accessible only within the scope of the `calculate_sum` function.

```
def calculate_sum(a, b):
    # Local variable
    result = a + b
    print("The sum is:", result)
```

```
The sum is: 8
```

```
# Calling the function
answer = calculate_sum(5, 3)
print("The sum is :",result)
```

```
NameError: name 'result' is not defined
```

'result' is only visible within the scope of the calculate_sum function

Check Your Understanding

What is the output of the following code?

```
1  def test():
2      x = 10
3      print("x is:", x)
4
5  x = 20
6  test()
7  print("x is:", x)
```

Check Your Understanding

What is the output of the following code?

```
1  def foo():
2      y = "Good day!"
3
4  foo()
5  print(y)
```

Check Your Understanding

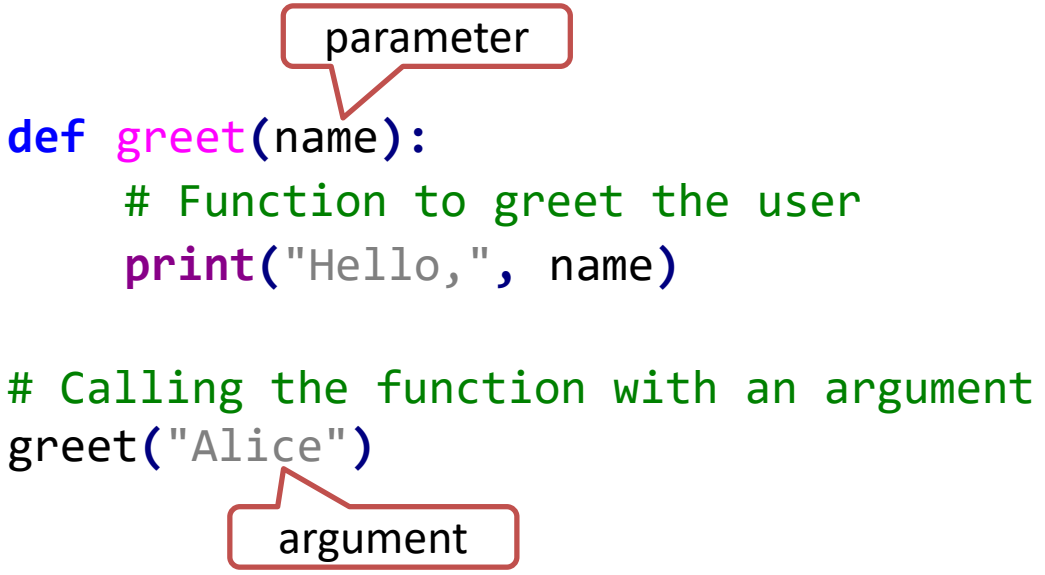
What is the output of the following code?

```
1  z = 10
2
3  def bar():
4      global z
5      z = z + 10
6
7  bar()
8  print(z)
```

Parameter Passing

Parameter Passing

- When we call a function, we can pass values to the function using parameters.
- **Parameters** are variables that act as placeholders for the values we want to pass to the function.
- The values passed to the function are called **arguments.**



Parameter Passing Modes

- In Python, there are two main parameter passing modes: pass by value and pass by reference.
- In **pass by value**,
  - a **copy of the variable's value** is passed to the function.
  - Any changes made to the parameter inside the function **do not affect** the original variable outside the function.

Pass by Reference

- Python uses a variation of pass by reference called "pass by assignment" or "pass by object reference"
- In pass by reference,
  - **the memory address of the variable** is passed to the function.
  - Any changes made to the parameter inside the function **affect the original variable** outside the function.

Immutable vs Mutable Objects

- Immutable objects, such as **numbers and strings**, are **passed by value**.

- Mutable objects, such as **lists, sets, and dictionaries**, are **passed by reference**.
  - *We'll discuss them later*

Check Your Understanding

What is the output of the following code?

```
1  def modify_value(x):
2      x += 5
3
4  value = 10
5  modify_value(value)
6  print(value)
```

Check Your Understanding

What is the output of the following code?

```
1  def modify_value(value):
2      value += 5
3
4  value = 10
5  modify_value(value)
6  print(value)
```

Exercises

[Recap] Computational Thinking

1. Decomposition: Breaking down complex problems into smaller, more manageable parts.
2. Pattern Recognition: Identifying patterns, similarities, and relationships within a problem.
3. Abstraction: Focusing on the essential details while ignoring irrelevant information.
4. Algorithm Design: Developing step-by-step instructions to solve a problem.

Algorithm Breakdown

- Breaking down the algorithm helps in understanding the problem and designing a modular and reusable solution.
- Algorithm breakdown involves breaking down a complex problem into smaller, more manageable steps.
- Each step should be well-defined and focused on solving a specific part of the problem.

Steps for Algorithm Breakdown

- Understand the Problem: Clearly define the problem and its requirements.
- Identify Inputs and Outputs: Determine the data or information required as input and the expected results as output.
- Break Down the Problem: Divide the problem into smaller sub-problems or tasks.
- Define Steps: For each sub-problem or task, define the specific steps needed to solve it.
- Sequence the Steps: Arrange the steps in a logical sequence, considering dependencies and order of execution.
- Test and Refine: Test the algorithm with sample inputs and refine it if necessary.

Example

- Problem: Calculate the square of a number.
- Algorithm Breakdown:
  1. Get the input number. (with the help of which Python built-in function?)
  2. Calculate the square of the number.
  3. Display the result. (with the help of which Python built-in function?)

Function Definitions:

```
def get_input_number():
    # Function to get the input number from the user
    # Code to retrieve and return the number

def calculate_square(num):
    # Function to calculate the square of a number
    # Code to perform the calculation
    # Return the square value

def display_result(result):
    # Function to display the calculated result
    # Code to display the result

#---------------------------------
#main ---------------------------------
number = get_input_number()
outcome = calculate_square(number)
display_result(outcome)
```

Scan for this coding exercise

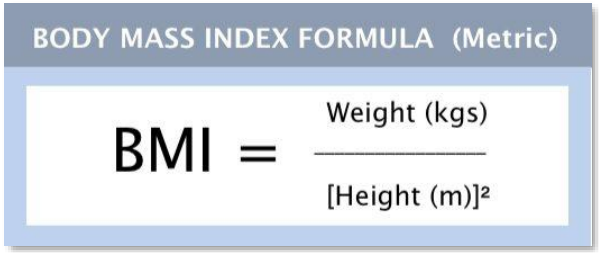
https://cmu.to/awXPC

Example

Problem: Body Mass Index Calculator (Input in cm and kg)

Algorithm Breakdown:

1. Get the input weight in kg from the user.
2. Get the input height in cm from the user.
3. Convert the height from cm to meters by dividing it by 100.
4. Calculate the square of the height.
5. Divide the weight by the square of the height to calculate the BMI.
6. Display the calculated BMI value.



Function Definitions: BMI Calculator

```
def get_weight():
    # Function to get the input weight in kg from the user
    # Code to retrieve and return the weight

def get_height():
    # Function to get the input height in cm from the user
    # Code to retrieve and return the height

def convert_to_meters(height_cm):
    # Function to convert height from cm to meters
    # Code to divide height_cm by 100 and return the result
```

Function Definitions: BMI Calculator

```
#Program BMI calculation
def get_user_input():
    #your code here
    return (input_w, input_h)

def calculate_bmi(in_weight, in_height):
    #your code here
    return bmi

def display_result(bmi):
    #your code here
#-------------------------------
#-------------------------------
#main function
weight, height = get_user_input()
bmi = calculate_bmi(weight, height)
display_result(bmi)
```

Scan for this coding exercise

https://cmu.to/Mbm1K

Summary

- A function is a section of reusable code.
- To call/invoke a function, type its name followed by a pair of parentheses.
- You have the option to pass data, known as arguments, to the function, but this requires a matching set of parameters in the function's definition.
- A function can return data back to the location where it was called.

References

- Hands-on Python tutorial
  - anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/functions.html
- Think Python: How to Think Like a Computer Scientist

W04 Lab

Understanding Function Call

Activity 1

- Form a group of 2-3 students
- Go to **play.blooket.com**
  - and enter Game ID [on screen]
- Answer the questions quickly and accurately to win the game
- After the game played, the statistics will be examined.

Visualize Python

- Python Tutor: Online python interpreter with visualization
  - https://pythontutor.com/visualize.html

Input as a String

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
known limitations

```
1  first = input("Input name: ")
2  last = input("Input lastname: ")
3  age = input("Input age: ")
4
5  print(first, type(first))
6  print(last, type(last))
7  print(age, type(age))
8
9  age_graduated = age + 4
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (7 steps)

TypeError: must be str, not int
(see KNOWN LIMITATIONS and unsupported features)

Print output (drag lower right corner to resize)

```
Input lastname: Smith
Input age: 17
Adam <class 'str'>
Smith <class 'str'>
17 <class 'str'>
```

Frames Objects

Global frame

first	"Adam"
last	"Smith"
age	"17"

https://tinyurl.com/29xsdk7f

Input conversion

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
known limitations

```
1  in_a = input("Enter a number :")
2  print(in_a, type(in_a))
3  a = int(in_a)
4  print(a, type(a))
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (4 steps)

Ads keep this tool free; we are not responsible for contents of displayed ads

Non-default options: show all frames (Python), primitive/object nesting, text labels for pointers

Move and hide objects

Print output (drag lower right corner to resize)

```
Enter a number :5
5 <class 'str'>
5 <class 'int'>
```

Frames Objects

Global frame

in_a	"5"
a	5

https://tinyurl.com/n3688syz

Input conversion

- Fix the code from https://tinyurl.com/29xsdk7f
- Make it run correctly

Print output (drag lower right corner to resize)

```
Input name: Adam
Input lastname: Smith
Input age: 17
Adam <class 'str'>
Smith <class 'str'>
17 <class 'str'>
21
```

Frames Objects

Global frame

first	"Adam"
last	"Smith"
age	"17"
age_graduated	21

What is the output of the following code?

- Type in below code in python tutor to visualize the see what it runs

```
1  def numval(num):
2      num += 1
3      print("num is: ",num)
4
5  value = 10
6  numval(value)
```

Activity 2

- Form a group of 2-3 students (Or you can do it soloist)
- Study the code in python tutor in order to understand the concept of parameter passing

Check Your Understanding

What is the output of the following code?

```
1  def test():
2      x = 10
3      print("x is:", x)
4
5  x = 20
6  test()
7  print("x is:", x)
```

Check Your Understanding

What is the output of the following code?

```
1  def modify_value(x):
2      x += 5
3
4  value = 10
5  modify_value(value)
6  print(value)
```

Check Your Understanding

What is the output of the following code?

```
1  def modify_value(value):
2      value += 5
3
4  value = 10
5  modify_value(value)
6  print(value)
```

Activity 3

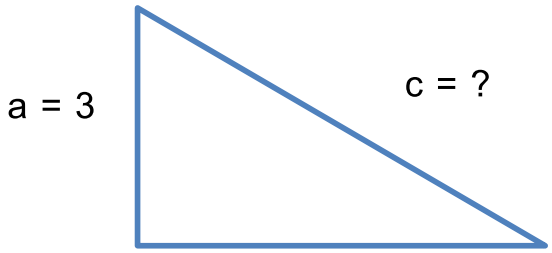
- Form a group of 2-3 students
- Go to play.blooket.com
  - and enter Game ID [on screen]
- Answer the questions quickly and accurately to win the game
- After the game played, the statistics will be examined

Coding Excercise

Write a Python program that calculates the length of the hypotenuse in a right triangle, given the lengths of the other two sides and the math module.

You are required to:

1. Create your own **square()** function that takes a number as an argument and returns its square.
2. Create a separate function to calculate the length of the hypotenuse in a right triangle. This function should take two arguments, each representing one side of the triangle.
3. The function created in step 2 should use the square() function from step 1, as well as the sqrt() function from **the math module**.
4. Ask the user for the lengths of the two sides of a triangle, pass the two lengths to the function in step 2.
5. The result should be printed using the format() function with the following text:



$$c^2 = a^2 + b^2$$
$$c = \sqrt[2]{a^2 + b^2}$$
$$= (a^2 + b^2)^{\frac{1}{2}}$$

"The hypothenuse in a right triangle with sides (**a**,**b**) is **xx.xx**"

xx.xx is a float value rounded to 2 decimal places

W05 Lab

Conditionals Part I

Sequential Program [Recap]

1. Prompt user for name
2. Read name
3. Prompt user for surname
4. Read surname
5. Prompt user for age
6. Read age

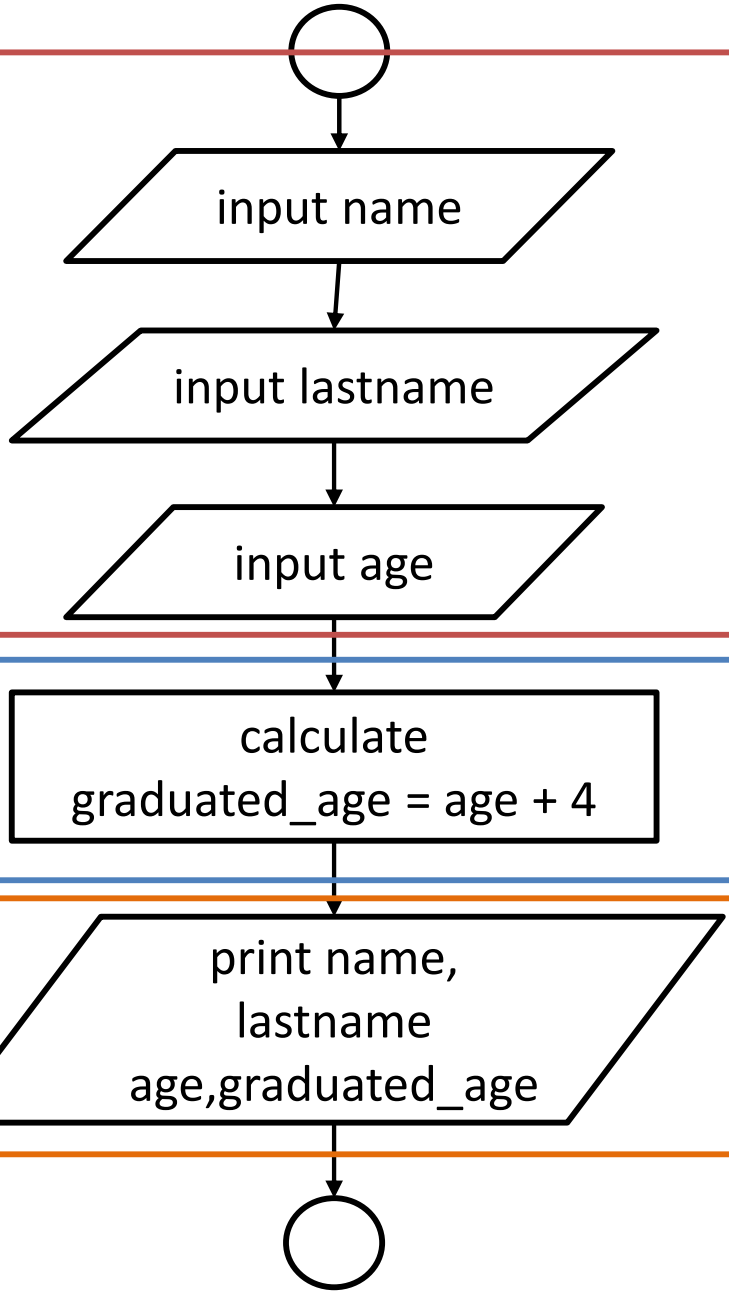
Input

7. Calculate graduation age (age + 4)

Process

8. Print name, surname, age, and graduation age

Output



Statement and Expression

- a statement
  - a line of code or a command that performs an action or operation
- Expression
  - a combination of values, variables, operators, and function calls
  - Can be evaluated to produce a result.
  - It represents a computation or a calculation.

```
x = 5
y = 3
result = x + y * (x - 2)
```

Boolean Expression

- Evaluate to either True or False.
- Used in conditions, loops, and other control flow statements.
- Involve relational operators and logical operators to compare values or combine conditions

Relational Operators

- Equality (==): Checks if two values are equal.
- Inequality (!=): Checks if two values are not equal.
- Greater than (>): Checks if one value is greater than another.
- Less than (<): Checks if one value is less than another.
- Greater than or equal to (>=): Checks if one value is greater than or equal to another.
- Less than or equal to (<=): Checks if one value is less than or equal to another.

Relational Operators

- Conditions are usually have comparisons
- Let a = 10 and b = 20

Operator	Description	Example	result
==	equal	a == b	
!=	not equal	a != b	
>	greater than	a > b	
<	less than	a < b	
>=	greater than or equal	a >= b	
<=	less than or equal	a <= b	

Check Your Understanding

- What will print out?

```
x = 5
y = 10
z = 7

r1 = x < y
r2 = z >= y
r3 = x != z

print(r1,r2,r3)
```

Common evaluation: is_even()

- Check if the number is even number (e.g 2, 4, 6, 8)
- Create function is_even(num)
  - takes an integer parameter num
  - returns True if the number is even and False if it is odd.
  - Inside the function, use the modulo operator (%) to check if num % 2 equals 0.

Common evaluation: is_even()

- Complete this code on editor and run

```
____ is_even(num):

    result = _________
    return_________
```

```
print(is_even(4))  #true
print(is_even(7))  #false
print(is_even(12)) #true
```

Voting Eligibility Check

- Write a function called `is_voting_age`
  - takes an integer parameter `age`
  - returns `true` if the person is of voting age (18 years or older)
  - returns `false` if they are not.
- Inside the function, use a boolean expression (age >= 18) to check if the age is greater than or equal to 18.
- If the condition is true, return True; otherwise, return False.
- Call the is_voting_age function with different test cases and print the returned value.

Common evaluation: is_voting_age()

- Complete this code on editor and run

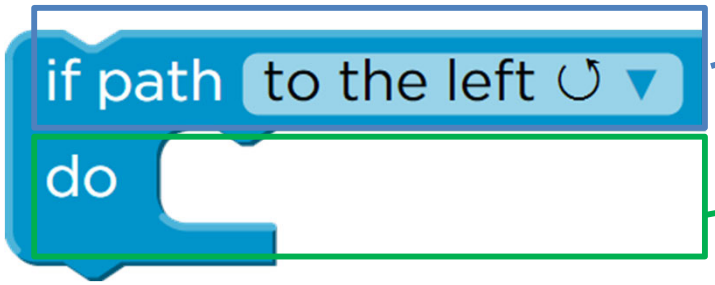
```
____ is_voting_age(age):

    result = _________
    return_________
```

```
print(is_voting_age(16))    #false
print(is_voting_age(21))    #true
print(is_voting_age(18))    #true
```

Condition Statement

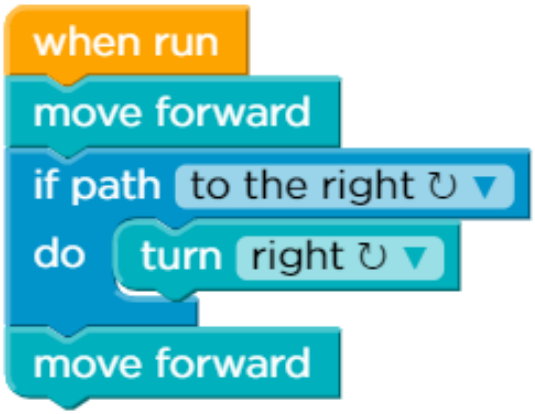
- Expression or a statement that evaluates to either true or false
- Conditions are used to control the flow and make decisions.
- Condition is used to determine which block of code should be executed based on the evaluated result

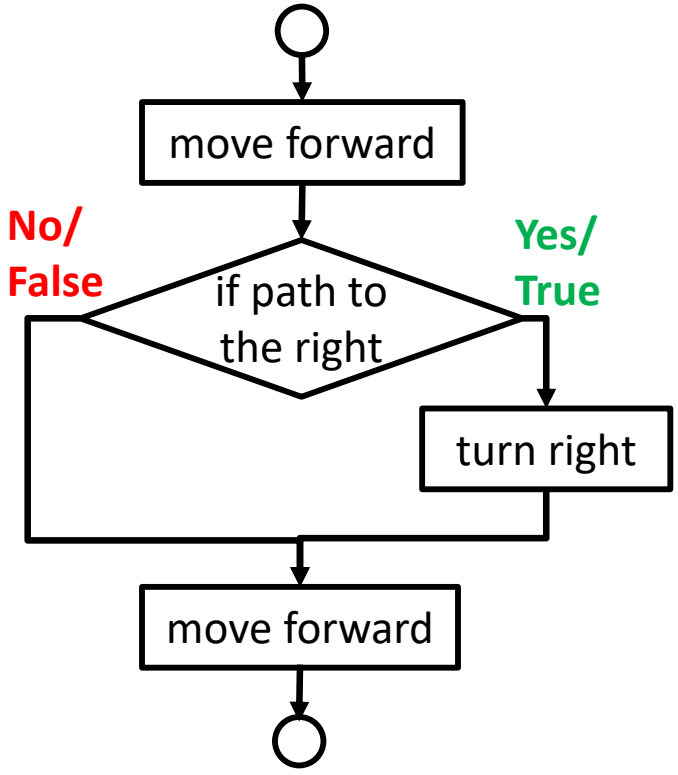
Condition Statement



Evaluate this expression

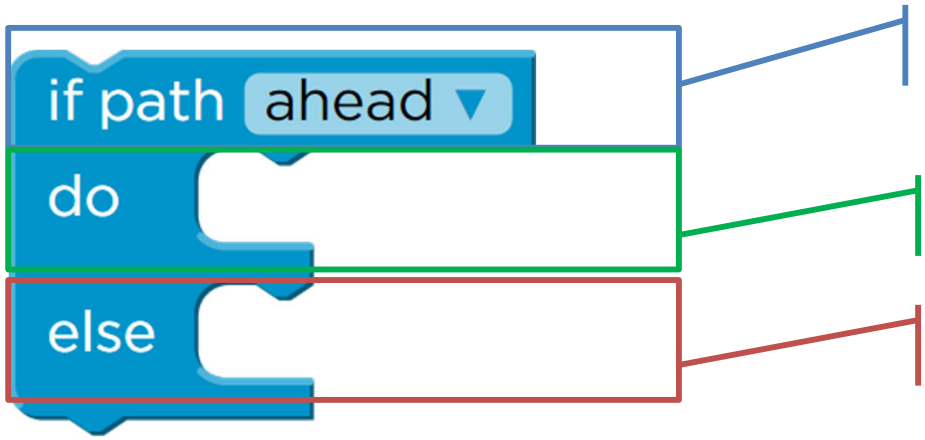
Do this when the expression is True






```
move_forward()
if  path_to_the_right:
        turn_right()
move_forward()
```

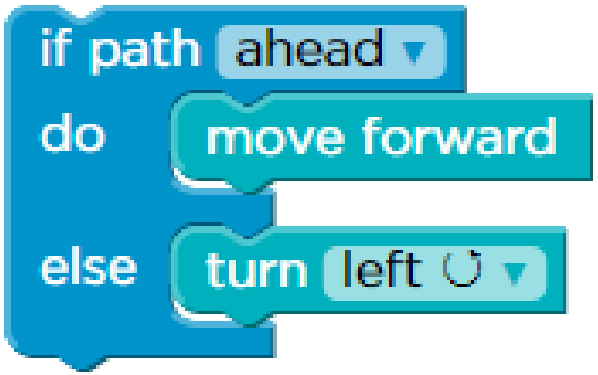
Condition Statement

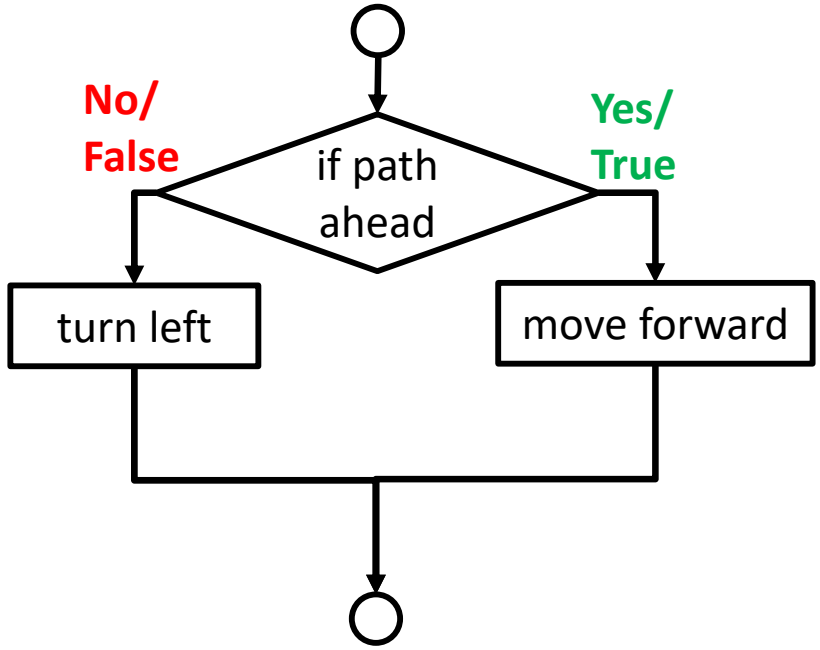


Evaluate this expression

Do this when the expression is True

Do this when the expression is False

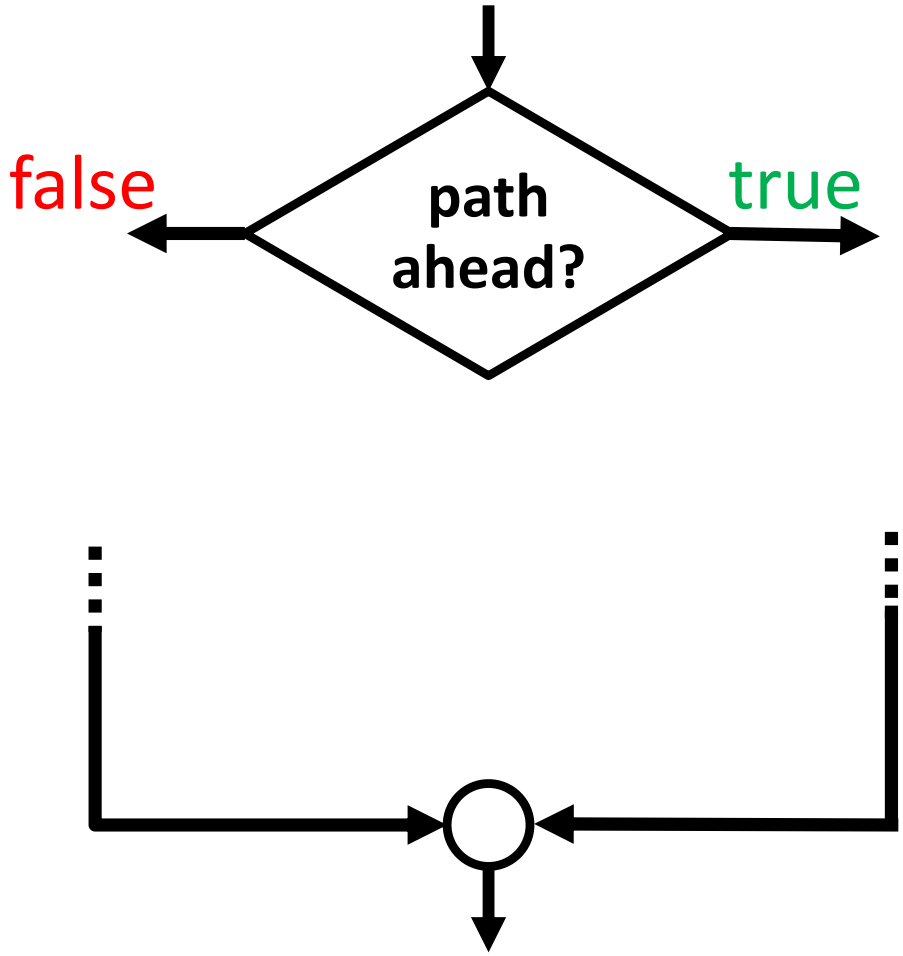





```
if path_ahead:
    move_forward()
else:
    turn_left()
```

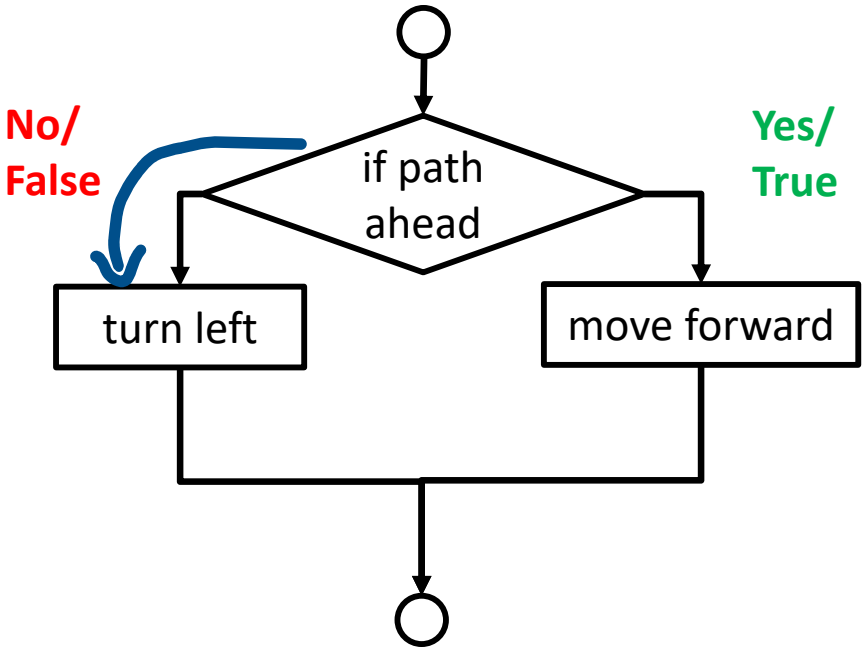
Decision Block

- Will allow the program to go on different path based on the condition provided.
  - If the condition is met (true), the program will continue on T- True path
  - If the condition is NOT met (false), the program will continue on F -False path instead.
- The paths may join back later on

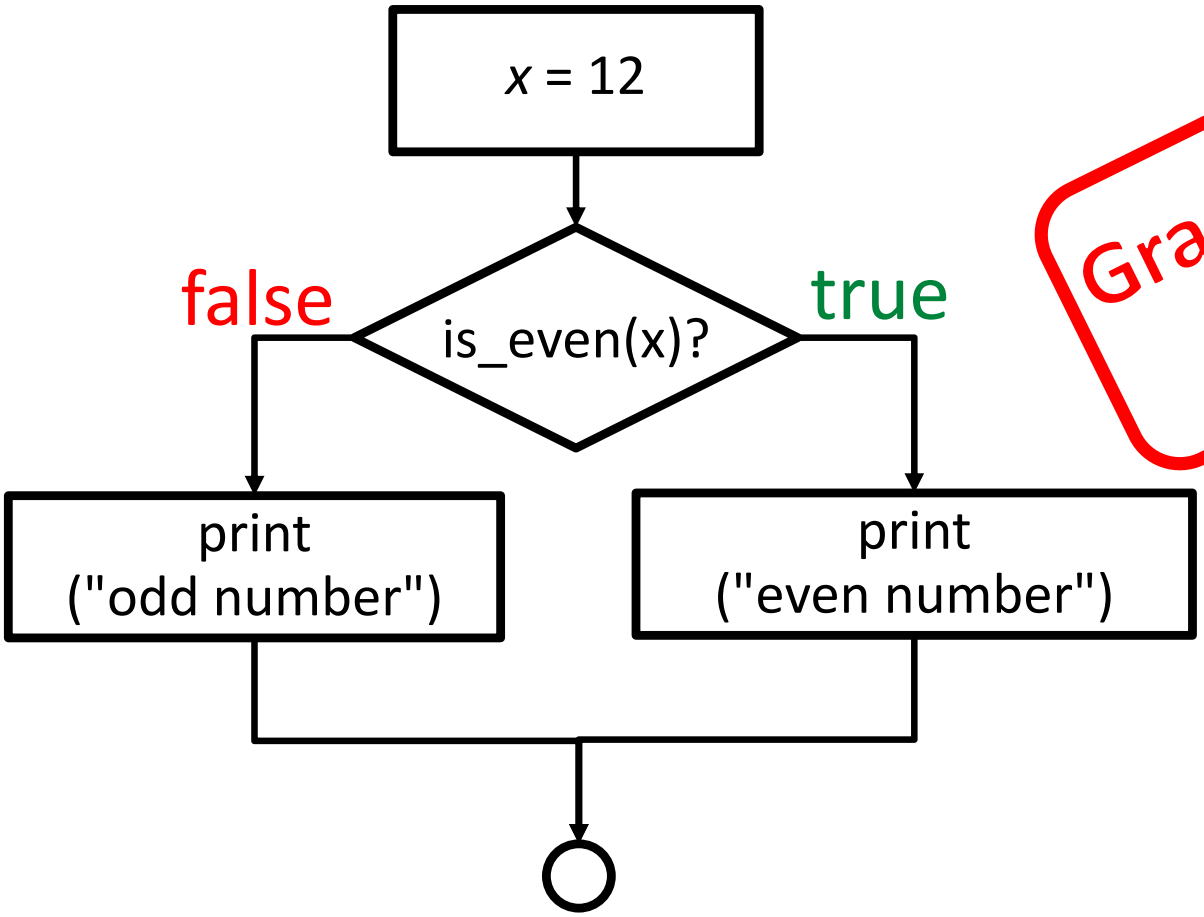


Selecting a Path

- Once a path is selected, the code/blocks on the other path will not be executed
- Unless looping is involved...



Conditional Flow control



Grader Alert!
w05_1

if statement

- if statement is a fundamental control structure in programming that allows the execution of certain code blocks based on a condition.

```
if condition:
    # code block to execute if the condition is true
```

```
x = 5
if x > 0:
    print("x is positive")
```

condition

don't forget :

run only if x > 0 is true

4 spaces indentation

if-else statement

- The if-else statement allows a program to execute different code blocks based on a condition.
- It provides an alternative path of execution when if statement evaluates to false.

```
if condition:
    # code block to execute if the condition is true
else:
    # code block to execute if the condition is false
```

```
x = 5
if x > 0:
    print("x is positive")
else:
    print("x is non-positive")
```

condition

4 spaces indent

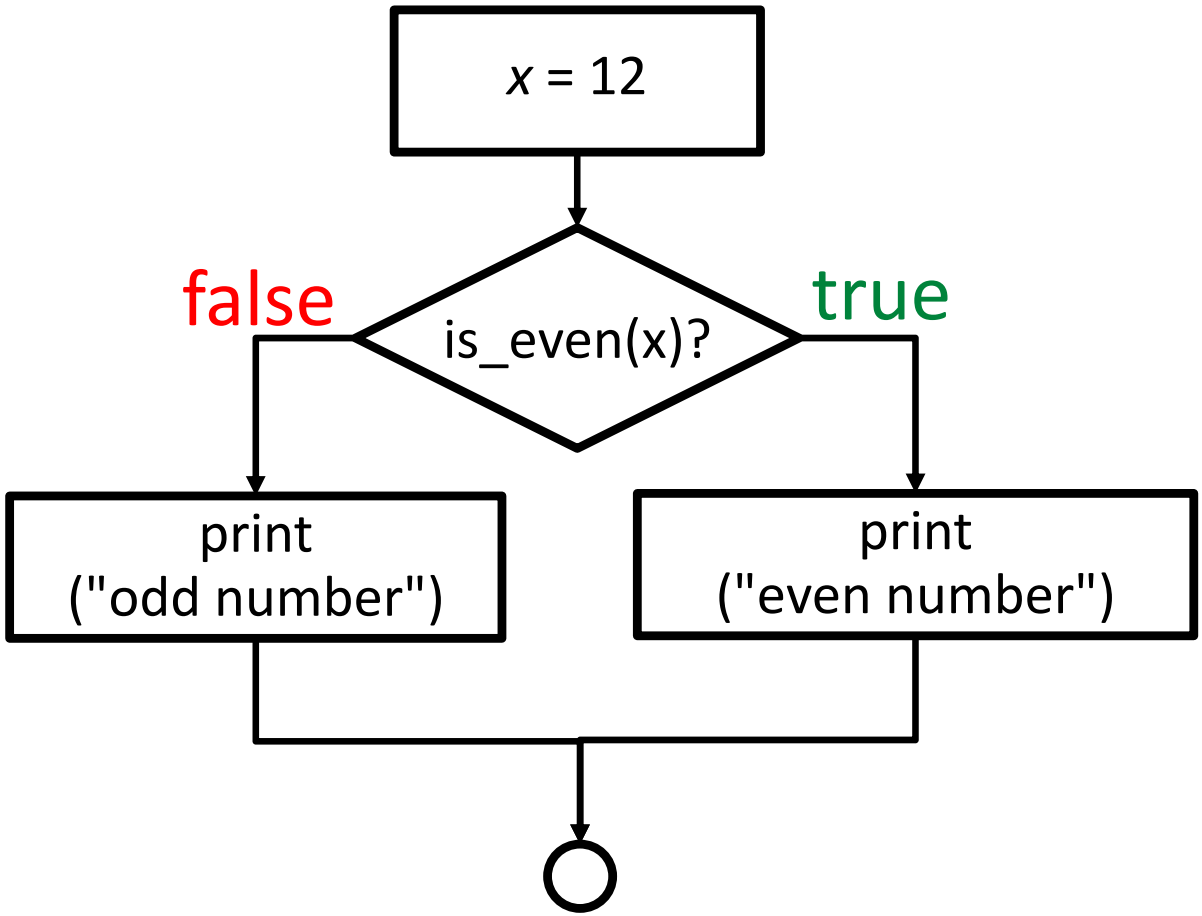
don't forget :

run only if x > 0

run when x ≤ 0 (condition is false)

Create Condition Flow with if-else

- Complete the code on the right hand side - reuse the is_even function for condition




```
def is_even(num):
…
…


x = 12
if _____________:
    print("odd number")
_____
    print("even number")
```

Create a function with Condition return

```
def function_name(parameters):
    # Code block
    return result
```

function definition

```
if condition:
    # code block to execute
    #if the condition is true
else:
    # code block to execute
    #if the condition is false
```

if-else condition

```
def function_name(parameters):
    if condition:
        result = "condition is true"
    else:
        result = "condition is false"
    return result
```

```
def function_name(parameters):
    if condition:
        return("condition is true")
    else:
        return("condition is false")
```

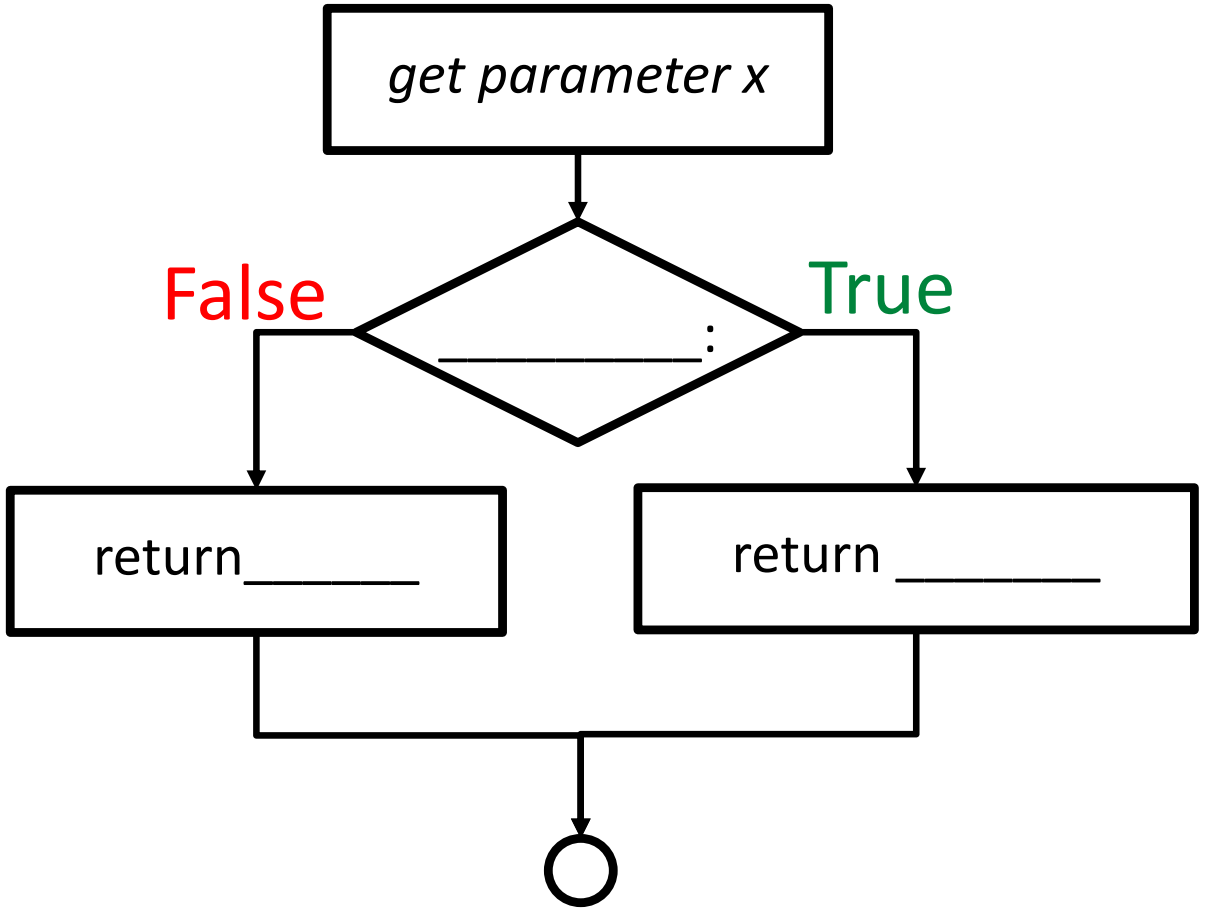
Create a function with Condition return

- The condition in a function can be based on various factors, such as the input parameters, intermediate calculations, or external variables.
- It allows the function to adapt its behavior and return different results based on different conditions.
- By using a function with condition return, you can encapsulate complex logic and decision-making within a reusable block of code.
- It promotes code reusability, modularity, and improves the overall readability and maintainability of your programs.

Create a function with Condition return

- Create a function named `is_positive` that takes 1 parameter: `num`.
- The function check if the parameter is positive or not
- return the result as Boolean value
  - True if num is positive number and return false if num Is negative number or zero

is_positive(num)



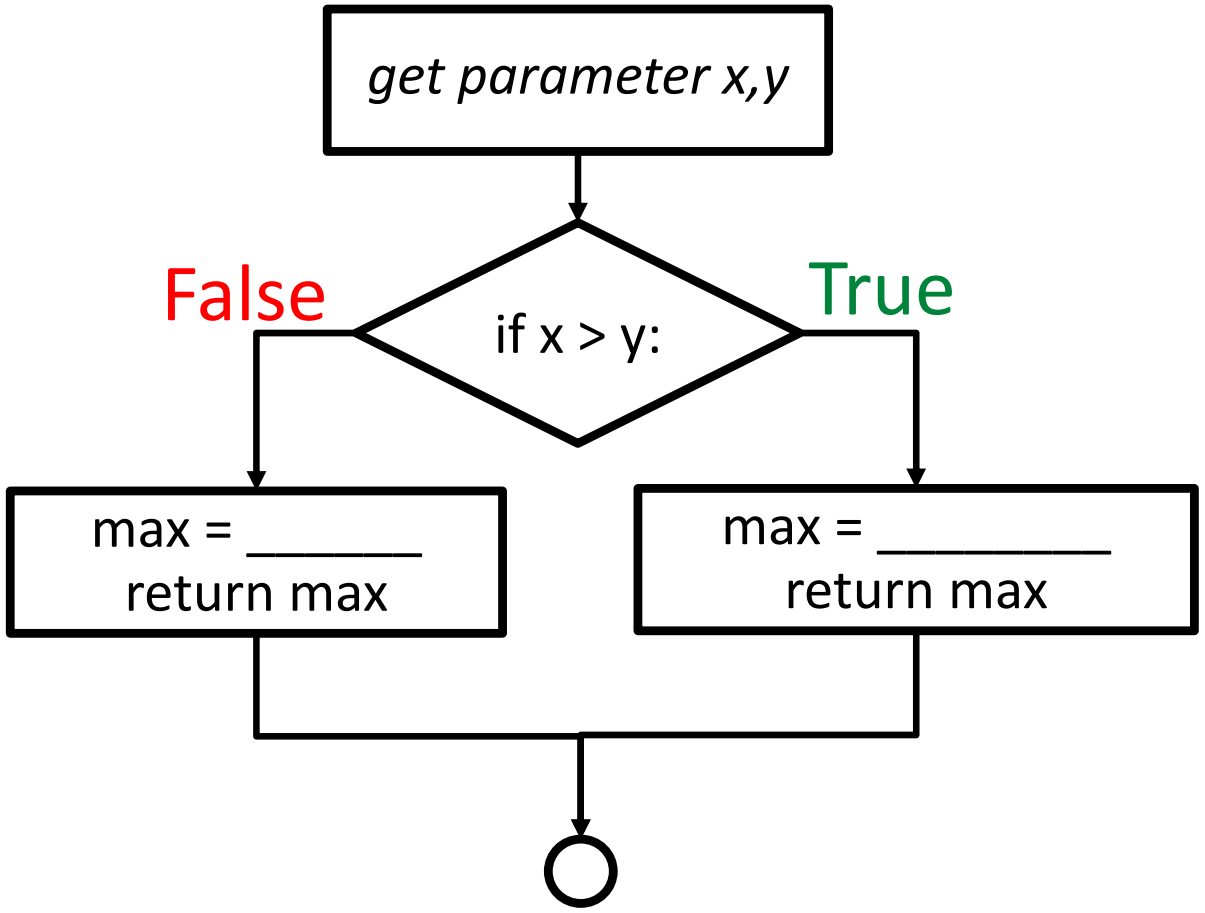
Grader Alert!
w05_2

```
def is_positive(num):
    if _____________:
        return _________
    else:
        return _________
```

```
print(is_positive(34))
print(is_positive(-12))
print(is_positive(0))
print(is_positive(75))
```

Function with Condition

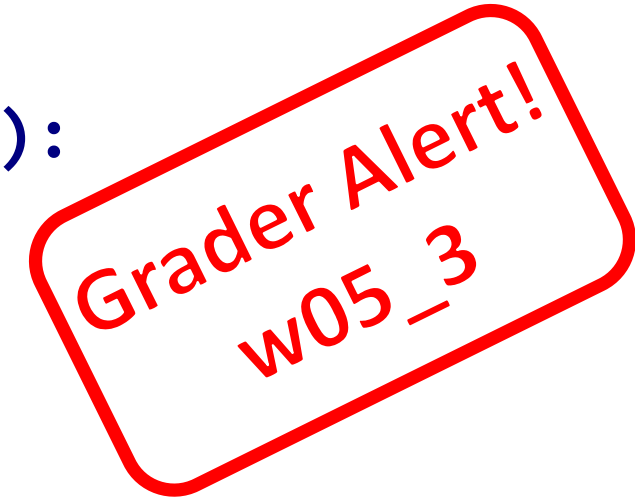
- Create a function that takes two numbers as input and returns the maximum of the two



Function with Condition

- Complete the function `find_max(num1,num2)`

```
def find_max(num1, num2):
if _________:
    return num1

_______
    return _____

result = find_max(5, 10)
print("The maximum number is:", result)
```



grader w05_3.py

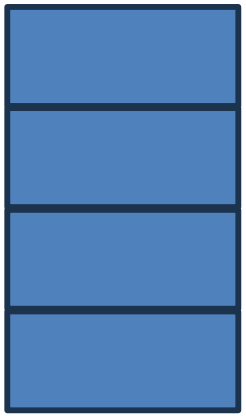
Logical Operators

- AND (and): Checks if both conditions are true.
- OR (or): Checks if either condition is true.
- NOT (not): Negates the result of a condition.

Logical Operators

- Logical operators (such as and, or) can join multiple Boolean variables/values/comparisons together
- For example, assuming

```
a = true, b = false and c = true
```

Operator	Example	Result
and	a and b	
	a and c	
or	a or b	
not	not(a and b)	

Multiple condition with Logical Operators

- Create more complex conditions to control the flow of your program.
- It will evaluate multiple conditions simultaneously
- Make decisions based on their combined results.
- Consider operator precedence and use parentheses () to ensure the desired evaluation order.

Multiple condition with Logical Operators

- If both conditions are true (i.e., the number is between 0 and 50), the code block indented under the if statement is executed, printing the message "The number is between 0 and 50."
- If either one or both conditions are false (i.e., the number is outside the range of 0 to 50), the code block indented under the else statement is executed, printing the message "The number is outside the range of 0 to 50."

both condition must be true

```
def check_range(number):
    if number >= 0 and number <= 50:
        print("The number is between 0 and 50.")
    else:
        print("The number is outside the range of 0 to 50.")

print(check_range(15))
print(check_range(72))
print(check_range(33))
```

indent for def

indent for if-else

Check Your Understanding

- What will print out?

```
x = 5
y = 10
z = 7

r1 = x < y
r2 = z >= y
r3 = x != z

print(r1,r2,r3)
print(r1 and r2)
print(r2 or r3)
```

Multiple condition with Logical Operators

- Checking if a person is eligible for a discount.
  - a person can get a discount whenever has a membership or age over 60 years old
  - write a function get_discount(age, has_membership)
    - age is integer
    - has_membership is Boolean

get_discount()

- Complete the code below

```
def get_discount_____________________________
    if ___________________________________:
        print("You are eligible for a discount.")
    else:
        print("You are not eligible for a discount.")
```

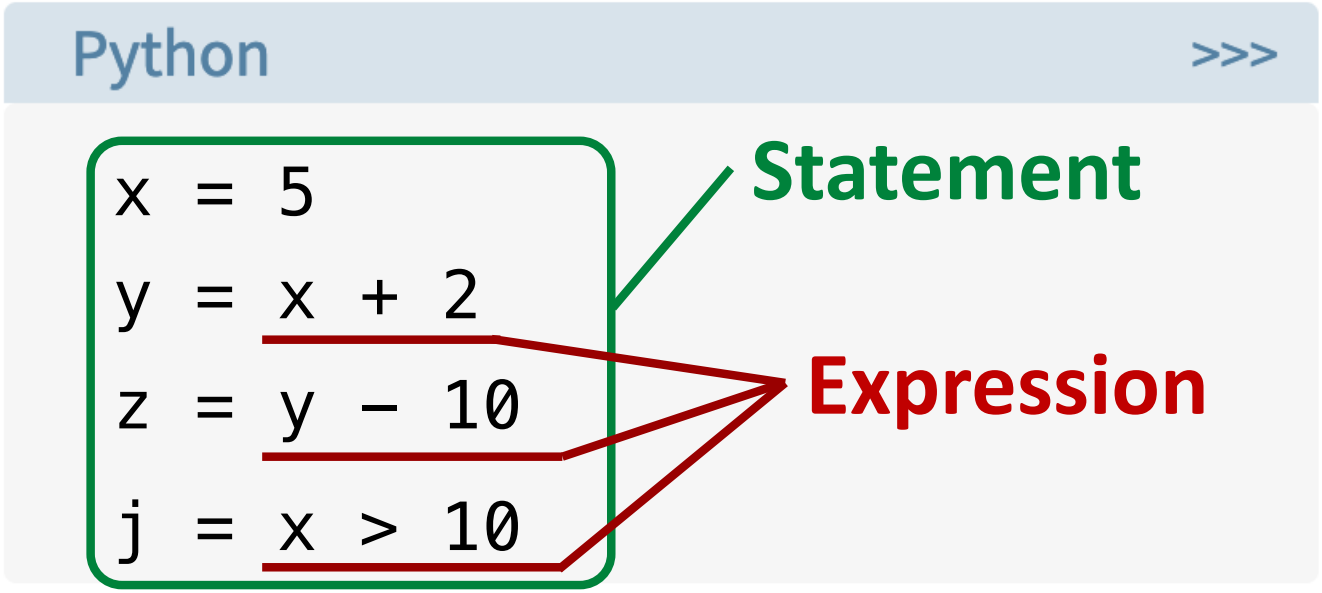
W05 LEC
Conditionals Part I

Agenda

- Statement vs Expression
- Boolean
- Boolean Operators
- Conditionals
  - If statement
  - If-else statement
- Exercises

Statement vs Expression

- **Statement** is a line of code
- **Expression** is a piece of code that **creates a value** or **evaluates to a value**.



Recap

- Three data types we have used so far are:
  - String (str) e.g., “Hello World”, “101”
  - Integer (int) e.g., 101
  - Floating-point (float), e.g., 3.14, 19.99

Today we are introducing a new data type:

Boolean

Boolean

Sounds like “Boo-Lee-Uhn”

IPA /ˈbuli.ən/

Boolean Data Type

- It has two possible values:
  - **True**
  - **False**

Note that ***True*** and ***False*** are reserved keywords in Python.

Python >>>

```
>>> type(False)
<class 'bool'>
>>> type(True)
<class 'bool'>
```

Boolean Data Type

- Like the other 3 data types we've used earlier, it is possible to assign a Boolean value to variables.
- But it is not possible to assign a value to True and False (because they are reserved keywords)

```
1  isHuman = True
2  print(isHuman)
3  print(type(isHuman))
```

Output:

```
True
<class 'bool'>
```

```
1  True = 5
```

→ Gives you a SyntaxError

Boolean as Numbers

- Boolean are considered a **numeric** type in Python. This means they are numbers for all intents and purposes.

True is 1

False is 0

Python >>>

```
>>> True == 1
True
>>> False == 0
True
>>> True + (False / True)
1.0
```

The bool() function

- Like the int() and float() functions, the bool() function converts a value to the corresponding Boolean value.

```
bool(1)
# result = true
```

```
bool("false")
# result = false
```

```
bool(0)
# result = false
```

```
bool("f")
# result = false
```

```
bool("1")
# result = true
```

```
bool("true")
# result = true
```

```
bool(6.66)
# result = true
```

```
bool("beneath the remains")
# result = true
```

Zero, in any numeric type such as int and float (e.g., 0, 0.0), will be evaluated to False. all other values will be True

Boolean Operators

Boolean Operators

- Boolean operators are those that take **Boolean inputs** and return **Boolean results.**
  - The "**not**" operator
  - The "**and**" operator
  - The "**or**" operator

The “not” operator

- This operator takes one argument and returns the opposite result.
- The “not” keyword is placed before Boolean variables/expressions.

Python >>>

```
>>> not True
False
>>> not False
True
```

p	~p
T	F
F	T

The “and” operator

- This operator takes two arguments.
- It only evaluates to True if both are True, otherwise it evaluates to False

Python >>>

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

The “or” operator

- This operator takes two arguments.
- It evaluates to True if one of the two arguments is True, or both are True.

p	q	p∨q
T	T	T
T	F	T
F	T	T
F	F	F

Python >>>

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

Comparison/Relational Operators

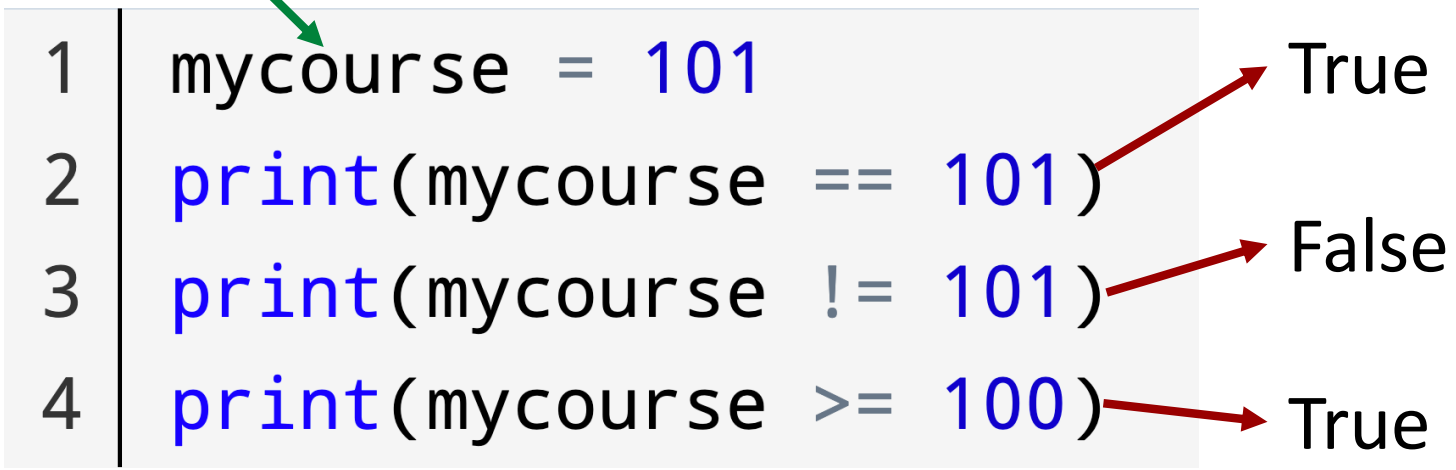
- Equality (==):
  - Checks if two values are equal.
- Inequality (!=):
  - Checks if two values are not equal.
- Greater than (>)
  - Checks if one value is greater than another.
- Less than (<)
  - Checks if one value is less than another.
- Greater than or equal to (>=)
  - Checks if one value is greater than or equal to another.
- Less than or equal to (<=)
  - Checks if one value is less than or equal to another.

These operators also take two arguments and **returns a Boolean**. These work for all 3 data types (string, float, int) we've learned so far

Example 1:

- The Boolean values ***True*** and ***False*** are returned when an expression is compared or evaluated with Comparison/Relational operators

Assigning 101 (int) to a variable named "mycourse"



Comparing the value stored in mycourse to an integer

Example 2:

- Let **a = 10** and **b = 20**

Comparing values stored in a and b in many different ways

Operator	Description	Example	Result
==	equal	a == b	
!=	not equal	a != b	
>	greater than	a > b	
<	less than	a < b	
>=	greater than or equal	a >= b	
<=	less than or equal	a <= b	

Combining Expressions

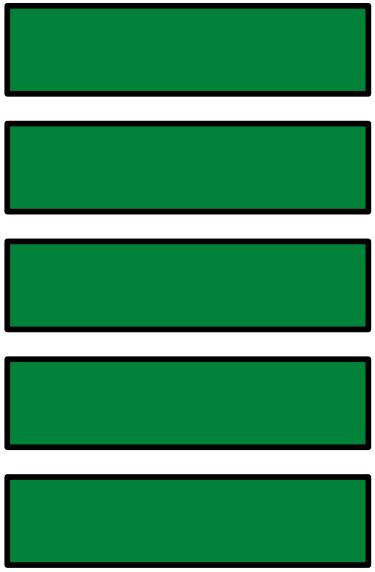
- You can then combine multiple expressions that evaluate to a Boolean to create more complex expressions

```
# Variables
temperature = 22
weather = "sunny"

# Multiple expressions that evaluate to a Boolean
is_warm = temperature > 20
is_sunny = weather == "sunny"

# Combine the expressions
should_picnic = is_warm and is_sunny

print(should_picnic) # Outputs: True
```

Check Your Understanding

Can you guess the outputs of the following code?

```
x = 10
y = 2
z = "hello101"
print(x == 5 or y < 5)
print((not x < 4) and y > 1)
print(x == 10 and y == 2)
print(z == "hello101" and y == 2)
print(z != "hello101" and y >= 2)
```



Be Careful

- It is easy to accidentally write an expression that is always True (***tautology***) or always False (***contradiction***)

What's wrong with these expressions?

```
count >= 10 or count < 50

count < 10 and count > 100
```

Conditionals

Control Structure

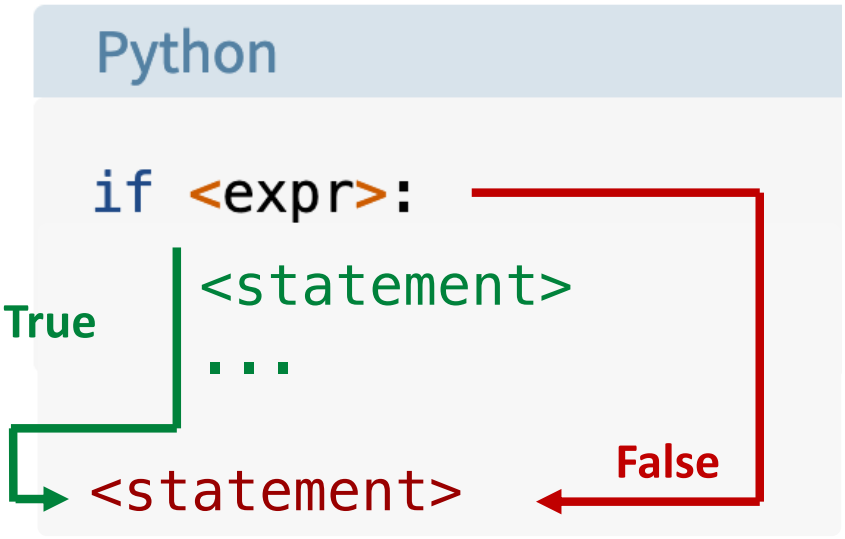
- So far we have only seen **sequential execution** (statements are performed one after the other)
- However, we may need to skip over some statements and perform some other statements based on some condition
- Ex. "*if it rains, I skip the class*."
- Ex. "*if I'm younger than 20 years old, I'm not legal to drink*"

Introducing
Control Structures

Control Structures

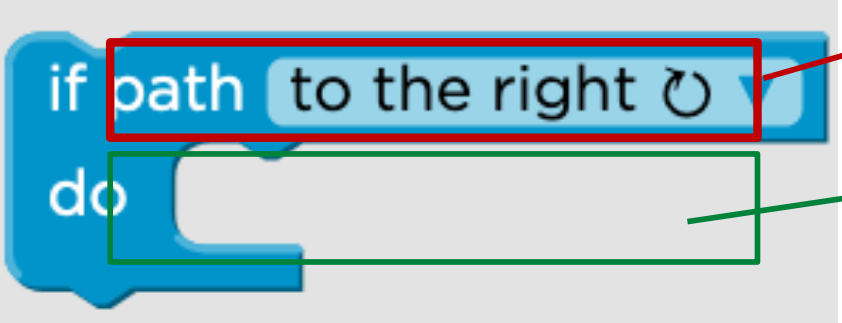
- A control structure directs the order of execution of the statements in a program.
- One of the basic control structures:

the `if` statement



- **<expr>** - an expression evaluated to a Boolean value we saw previously.
- **<statement>** a block of statements (**note the identation)** that will be executed when the <expr> is evaluated to **True** or skip over all **<statements>** if **False**
- Execution will then proceed with <statement> either way

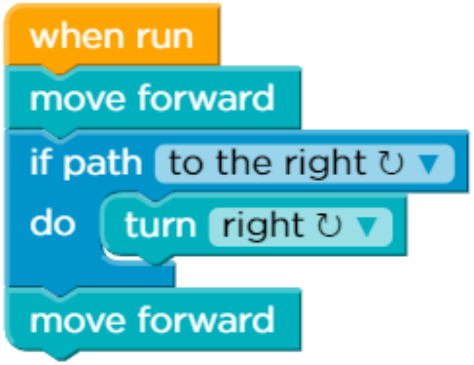
Remember the Maze Game?



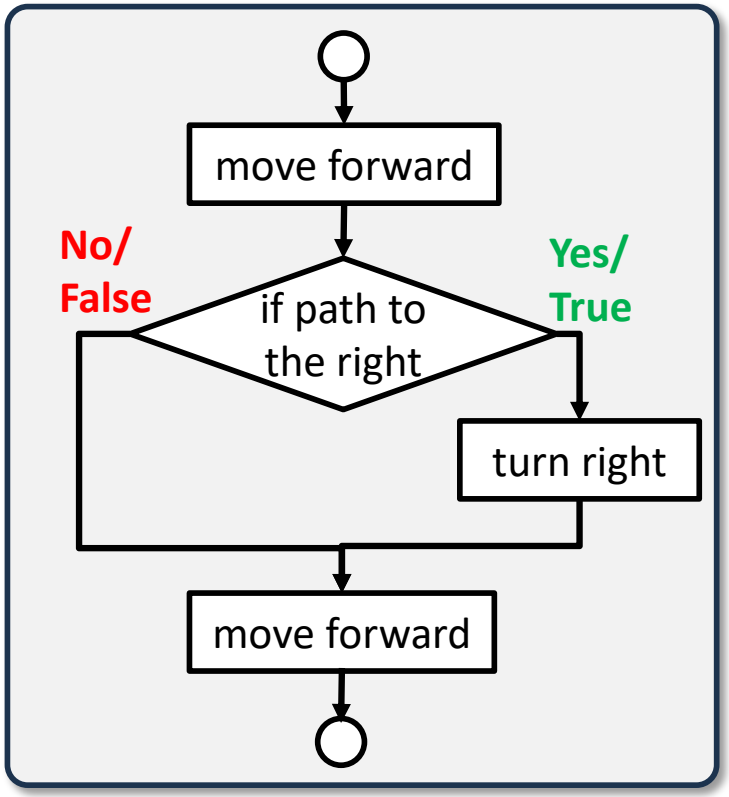
Evaluate this expression

Do this when the **expression** is True

Block-based View



Flow Chart View



Python Code View

```
move_forward()
if  path_to_the_right:
        turn_right()
move_forward()
```

Check Your Understanding

Can you guess the output?

```
x = 10
if x == 10:
    print("the expr is True")
    print("this block is executed")

print("after conditional")
```

Answer

```
x = 10
if x != 10:
    print("the expr is True")
    print("this block is executed")

print("after conditional")
```

Answer

Coding Excercise

- Write a program to prompt the user for a number
- If the user enters any positive number, print "*Good job!*" *and* "*You guessed it right*" on two separately lines
- Regardless of whether the user guessed the number correctly, the program should print "*Thanks for playing the game!*"

Coding Exercise - Solution

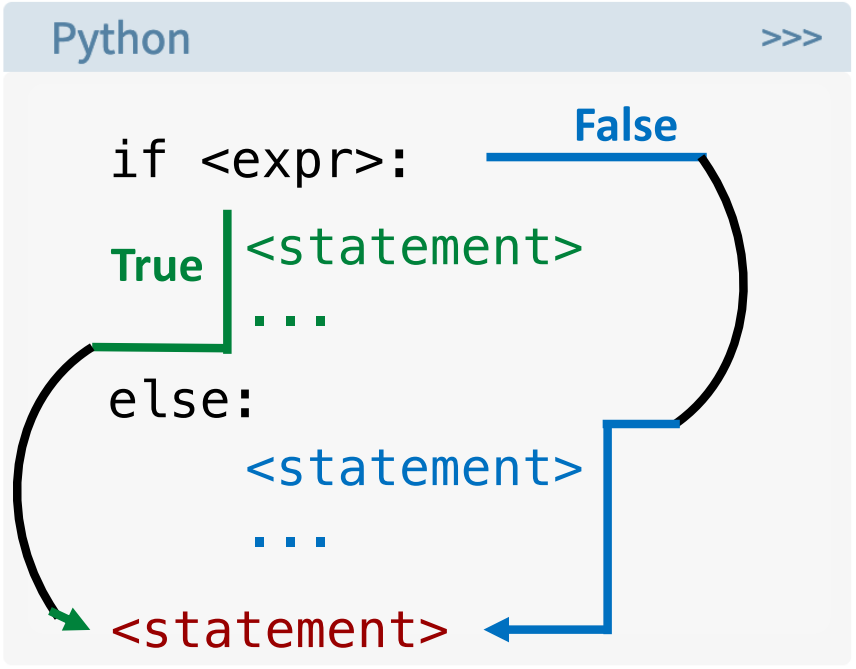
Solution

A More Advanced Control Structure

Sometimes, you want to evaluate a condition and take on path if its true but specify an alternative path if it is not.

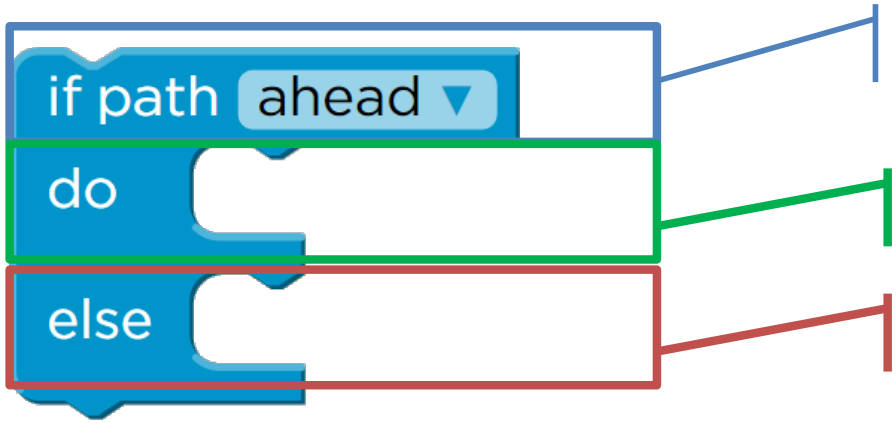
The `if-else` statement

Python >>>



- **<expr>** - an expression evaluated to a Boolean value we saw previously.
- **<statement>** a block of statements that will be executed if <expr> is evaluated to **True**
- **<statement>** a block of statements that will be executed if <expr> is **False**
- Execution will then proceed with **<statement>** either way

Remember the Maze Game?

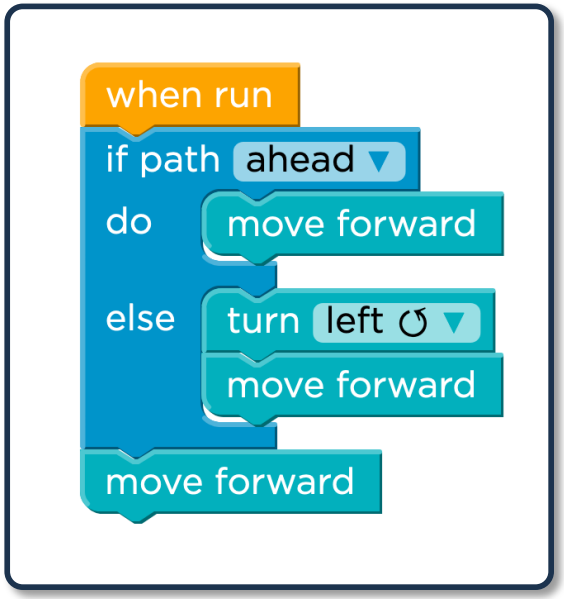


Evaluate this expression

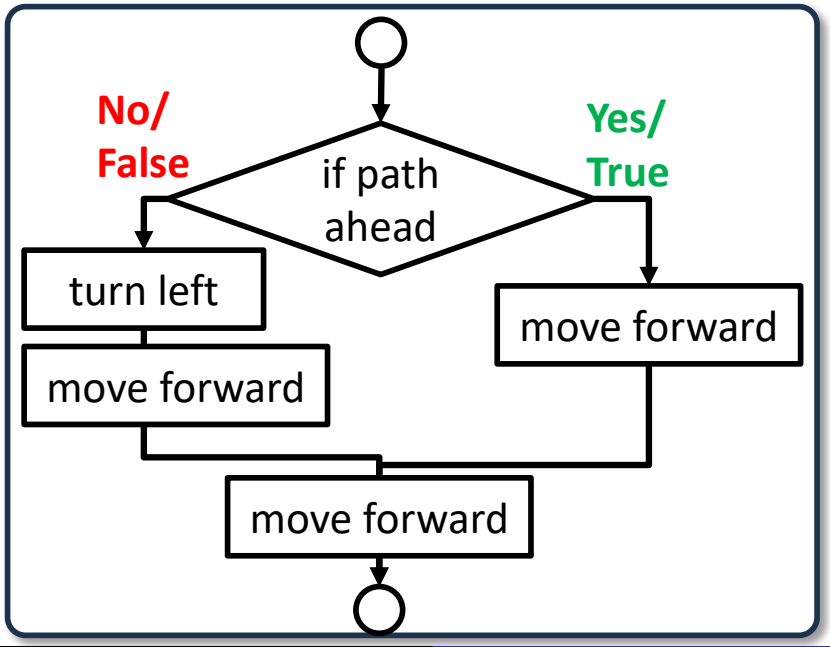
Do this when the expression is True

Do this when the expression is False

Block-based View



Flow Chart View

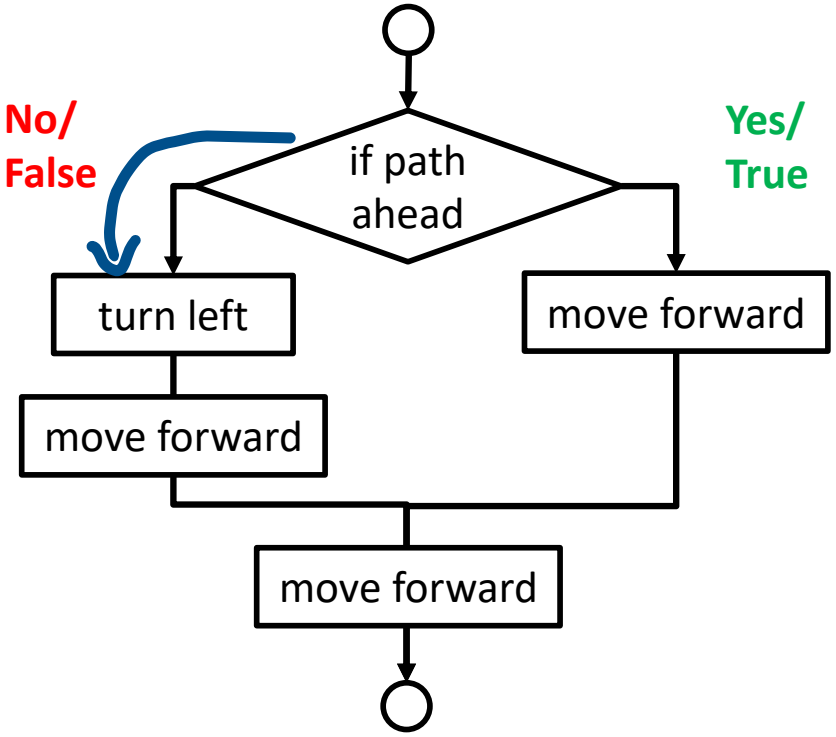


Python Code View

```
if path_ahead:
    move_forward()
else:
    turn_left()
    move_forward()
move_forward()
```

Selecting a Path

- Once a path is selected, the code/blocks on the other path will not be executed
- Unless looping is involved...



Real Life Scenario

If the weather is "nice"

- Take my dog for a walk
- Ride a bicycle
- Go grocery shopping

else

- Stay at home
- Watch Netflix Movies/TV Shows
- Have a facetime call with friends

Regardless, **I'll have dinner at home**

Real Life Scenario

If the weather is "nice"

- Take my dog for a walk
- Ride a bicycle
- Go grocery shopping

else

- Stay at home
- Watch Netflix Movies/TV Shows
- Have a facetime call with friends

Regardless, I'll **have dinner at home**

Let's turn this into code

(print out each action on separate lines)

One requirement though

You have to prompt the user to enter the condition of the weather.

(Hint: see how you can compare strings in Slide #19)

Real Life Scenario to Code

Solution

Coding Exercises

Exercise – 1 (Clubbing)

As a bouncer at a club, your task involves deciding who can enter. Specifically, you need to verify if a person is at least 21 years old. If they meet this age requirement, you allow them entry. If not, you deny them entry.

Your program should ask the user for their age.

Exercise – 2 (Finding the min value)

Write a function named "*find_min*" that takes in two numbers as arguments and return the minimum of the two.

(The function should not have any print statement)

You should ask the user to input the two numbers

Exercise – 3 (Am I eligible to vote?)

Write a program that asks the user for their age and asks whether they are a citizen of the country ("yes" or "no").

If the user is 18 years old or older and they are a citizen, the program should print "You are allowed to vote." Otherwise, the program should print "You are not allowed to vote."

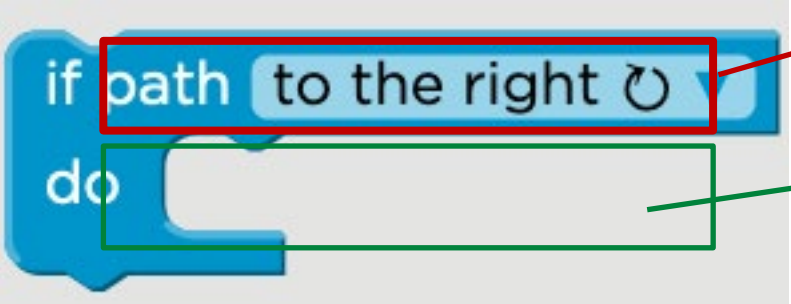
References:

- https://realpython.com/python-conditional-statements/
- https://realpython.com/python-boolean/

W06 LAB

Multiple Conditions

Remember the Maze Game? [recap]

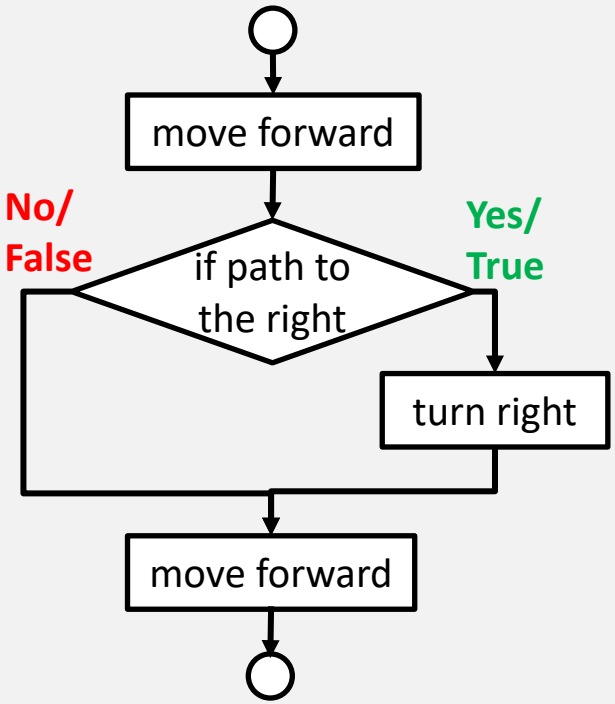


Evaluate this expression

Do this when the **expression** is True

Block-based View

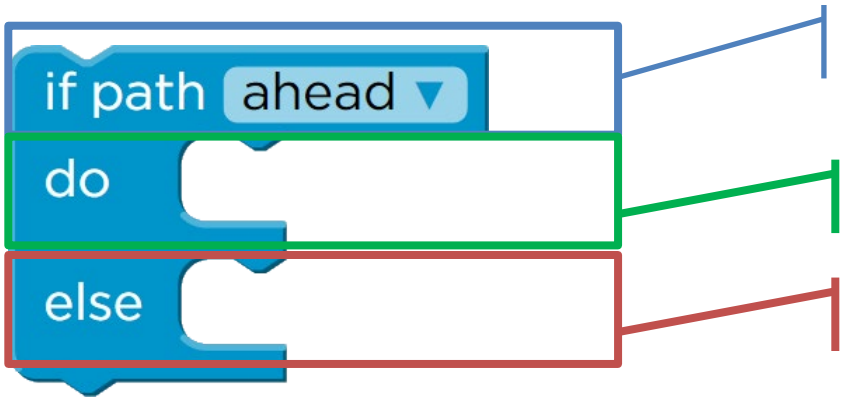
Flow Chart View



Python Code View

```
move_forward()
if path_to_the_right:
    turn_right()
move_forward()
```

Remember the Maze Game? [recap]



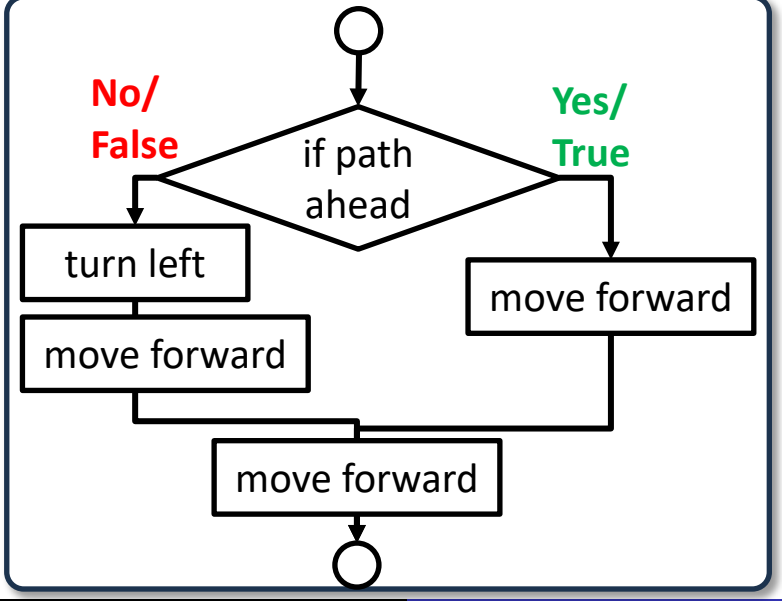
Evaluate this expression

Do this when the expression is True

Do this when the expression is False

Block-based View

Flow Chart View

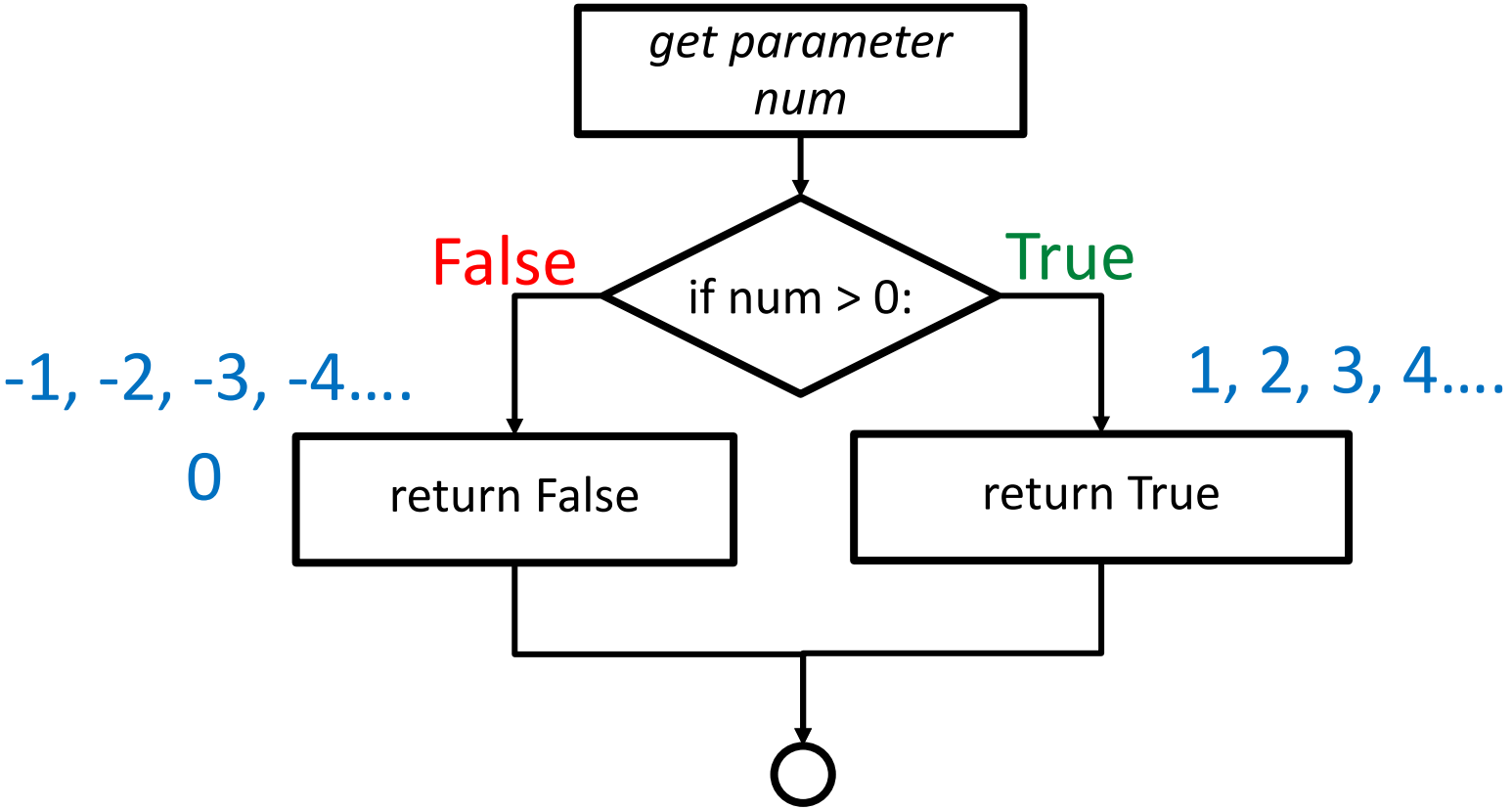


Python Code View

```
if path_ahead:
    move_forward()
else:
    turn_left()
    move_forward()
move_forward()
```

Chain Conditionals

- reconsider is_positive(num)

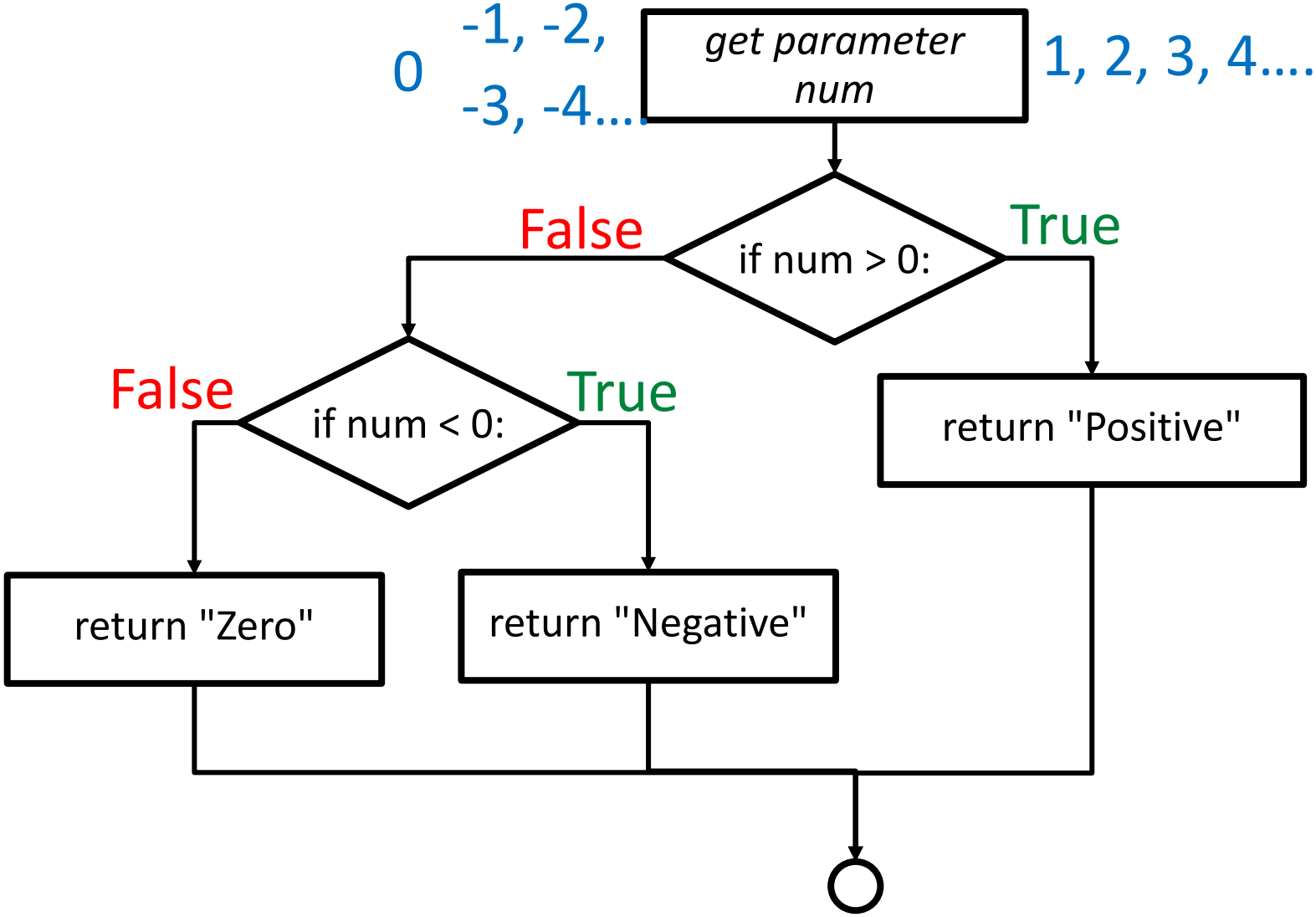


what if we want to separate positive, zero, negative number?

Multiple conditions

- the situation where more than one condition needs to be evaluated to determine the outcome or behavior of a program.
- It involves checking multiple expressions or variables against specific criteria and executing different code blocks based on the results of those conditions.

positive, zero, negative number?



Multiple Condition Problems

- Question: Discount Calculation
  - If the total purchase amount is less than $50, there is no discount.
  - If the total purchase amount is between $50 and $100 (inclusive), apply a 10% discount.
  - If the total purchase amount is greater than $100, apply a 20% discount.

Multiple Condition Problems

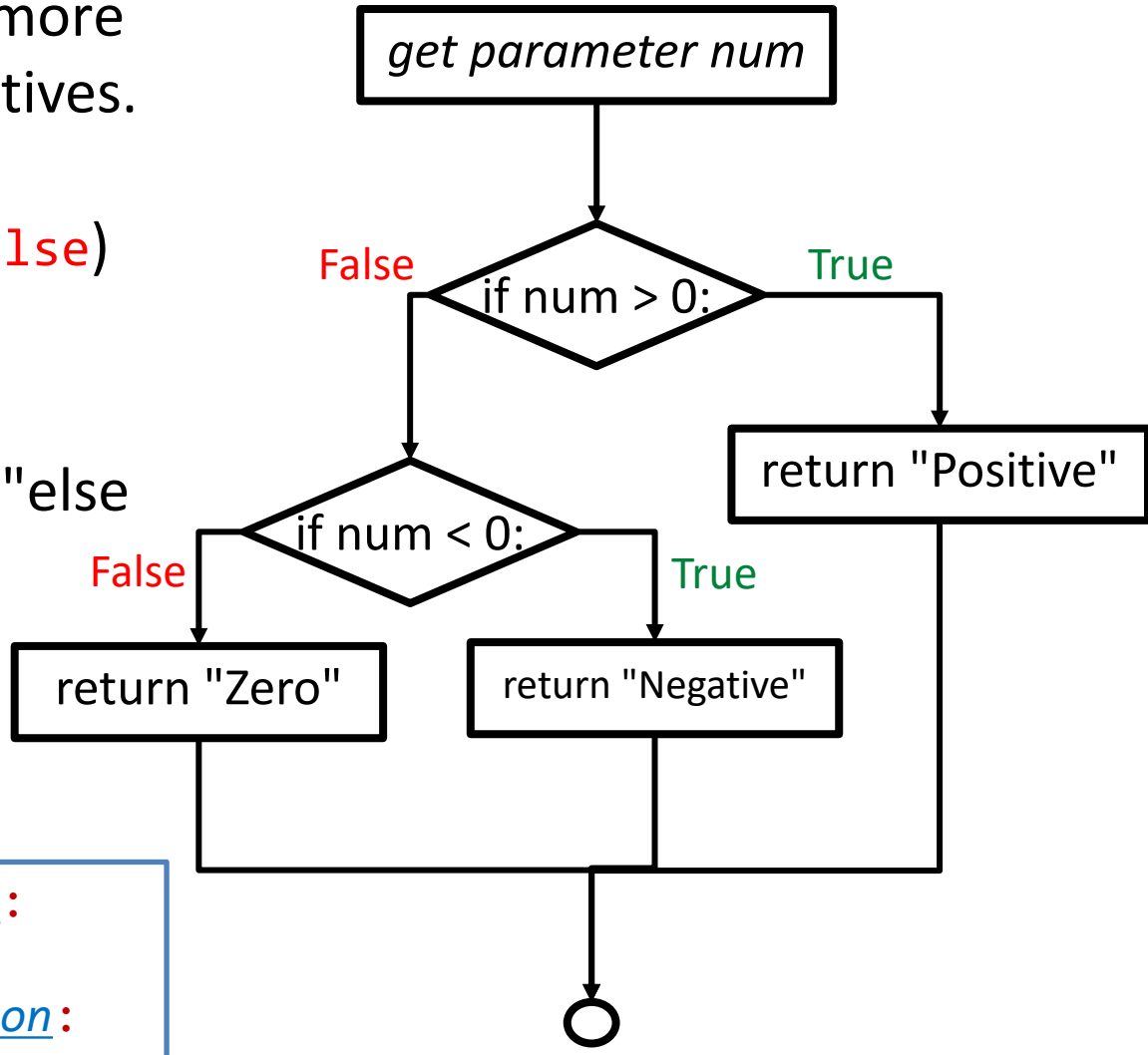
- Question: Ticket Pricing A theater has different ticket prices based on the age of the person.
  - Children (age 0-12): $5
  - Teenagers (age 13-18): $10
  - Adults (age 19-64): $15
  - Seniors (age 65 and above): $12

Multiple Condition Problems

- Question: Student Grading
  - If the score is equal to or greater than 90, it prints "Excellent!"
  - If the score is between 80 and 89 (inclusive), it prints "Good job!"
  - If the score is between 70 and 79 (inclusive), it prints "Keep it up!"
  - If the score is between 60 and 69 (inclusive), it prints "You can do better."
  - If the score is below 60, it prints "You need to study harder."

Chained Conditionals

- In some cases, there are more than two possible alternatives. We can use a chained condition (`if - elif - else`) to support this decision condition.
- elif is an abbreviation for "else if".
- from all possible options only one instruction set will be executed.

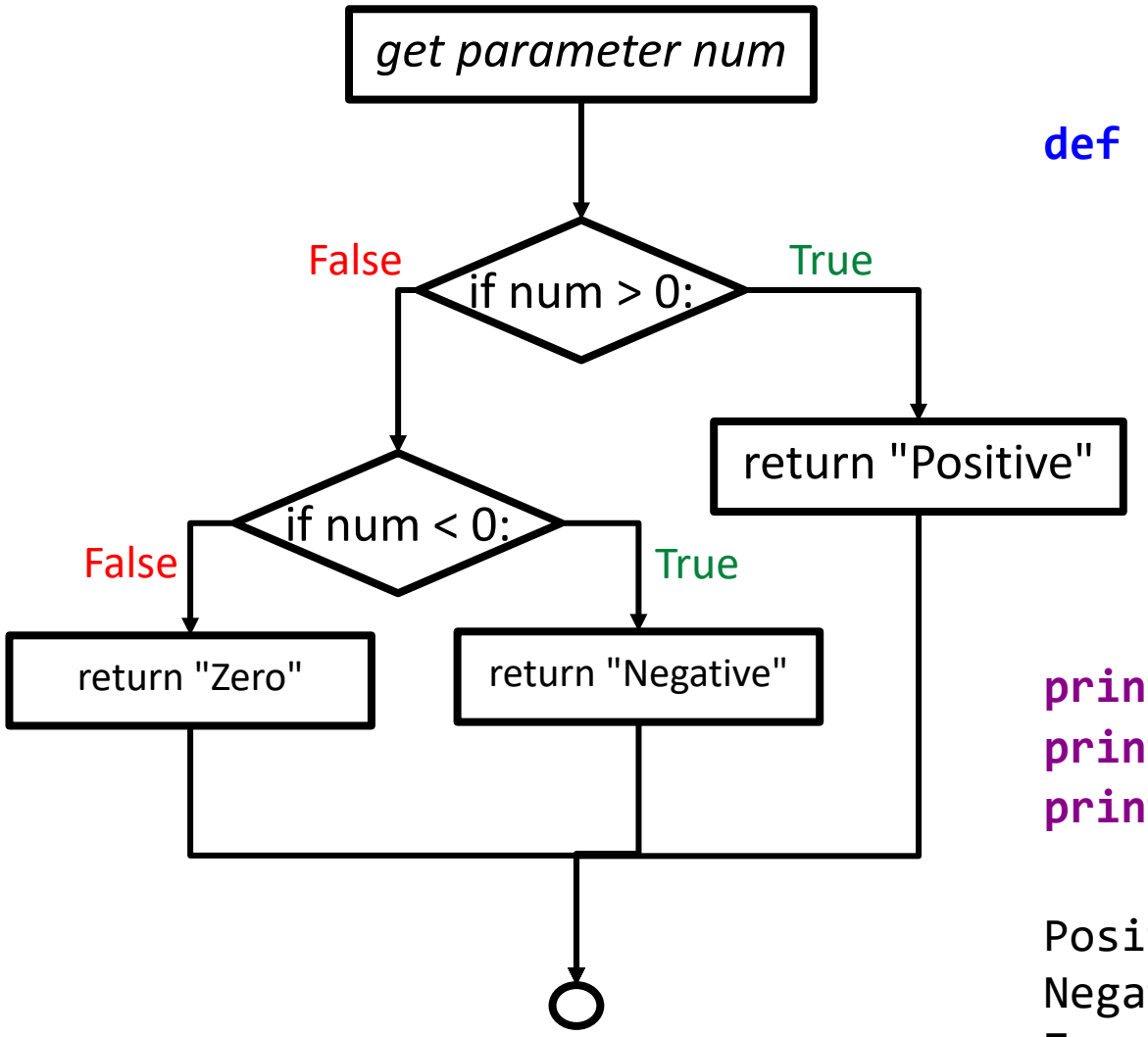



```
if Boolean expression:
    block of code
elif Boolean expression:
    block of code
else:
    block of code
```

elif (else if) Statement

- elif statement is used after if statement for further decision.
- Code block under elif statement will be executed when:
  1. The condition for the if statement (and the ones for elif statement above this one) is not met, and
  2. The condition for the current elif is met

Code vs. Flowchart




```
def int_type(num):
    if num >0:
        return "Positive"
    elif num < 0:
        return "Negative"
    else:
        return "Zero"
```

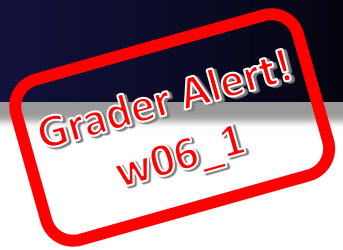
```
print(int_type(5))
print(int_type(-8))
print(int_type(0))
```

```
Positive
Negative
Zero
```

Example – Ticket Drawing

- If you randomly select a number between 1 and 99, the following outcomes are possible:
  - If the number falls between 45 and 55 (inclusive), you win the first prize.
  - If the number falls between 15 and 30 (inclusive) or between 75 and 90 (inclusive), you win the second prize.
  - For any other number, there is no prize.

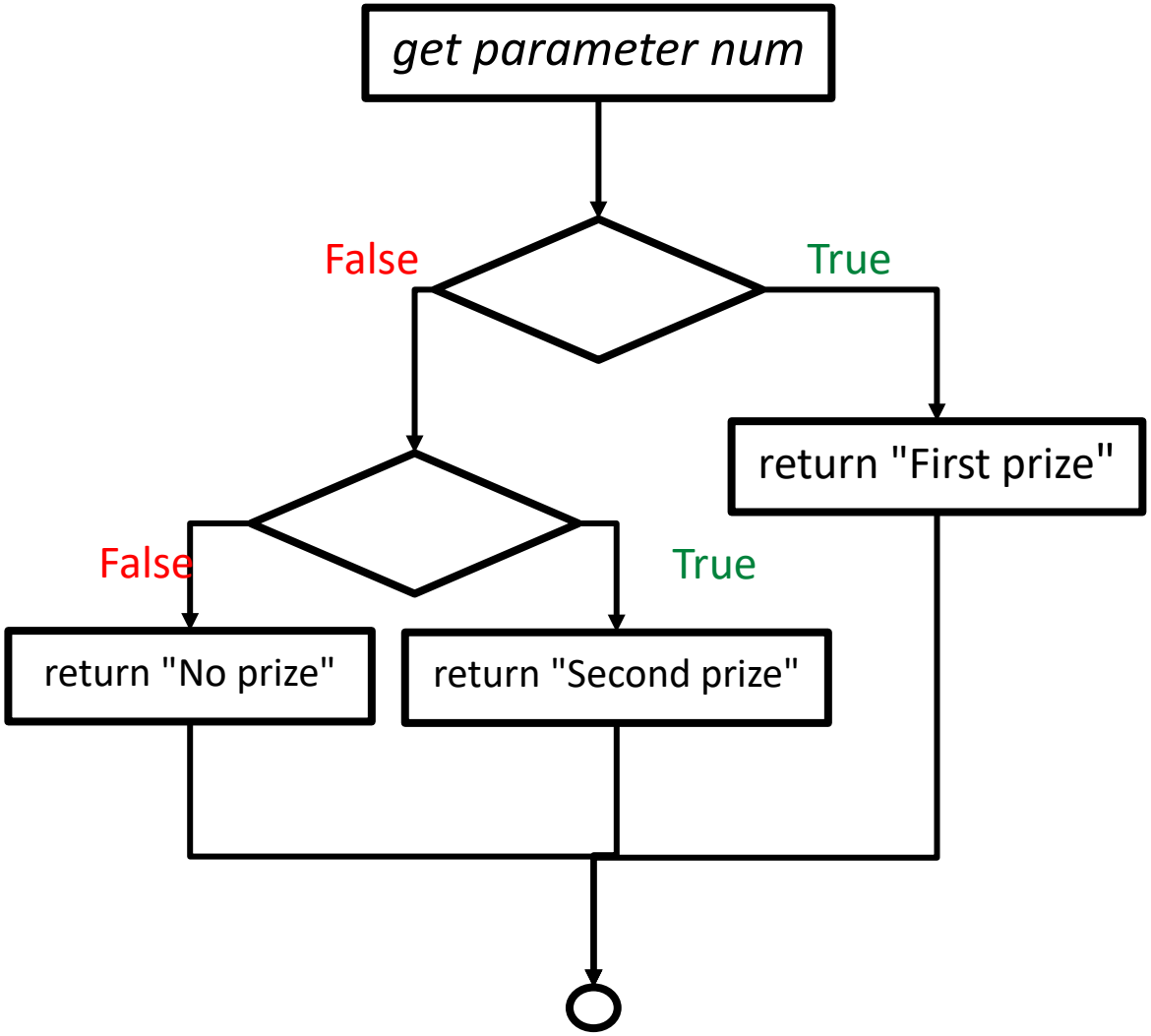
Ticket Drawing



- Break down the conditions for num
  - between 45 and 55 (inclusive) -first prize
    - Boolean expression is: 45 <= num <=55
  - between 15 and 30 (inclusive) - second prize
    - Boolean expression is:_____
  - between 75 and 90 (inclusive) - second prize
    - Boolean expression is:_____
  - other number - no prize
    - Boolean expression is:_____

Ticket Drawing

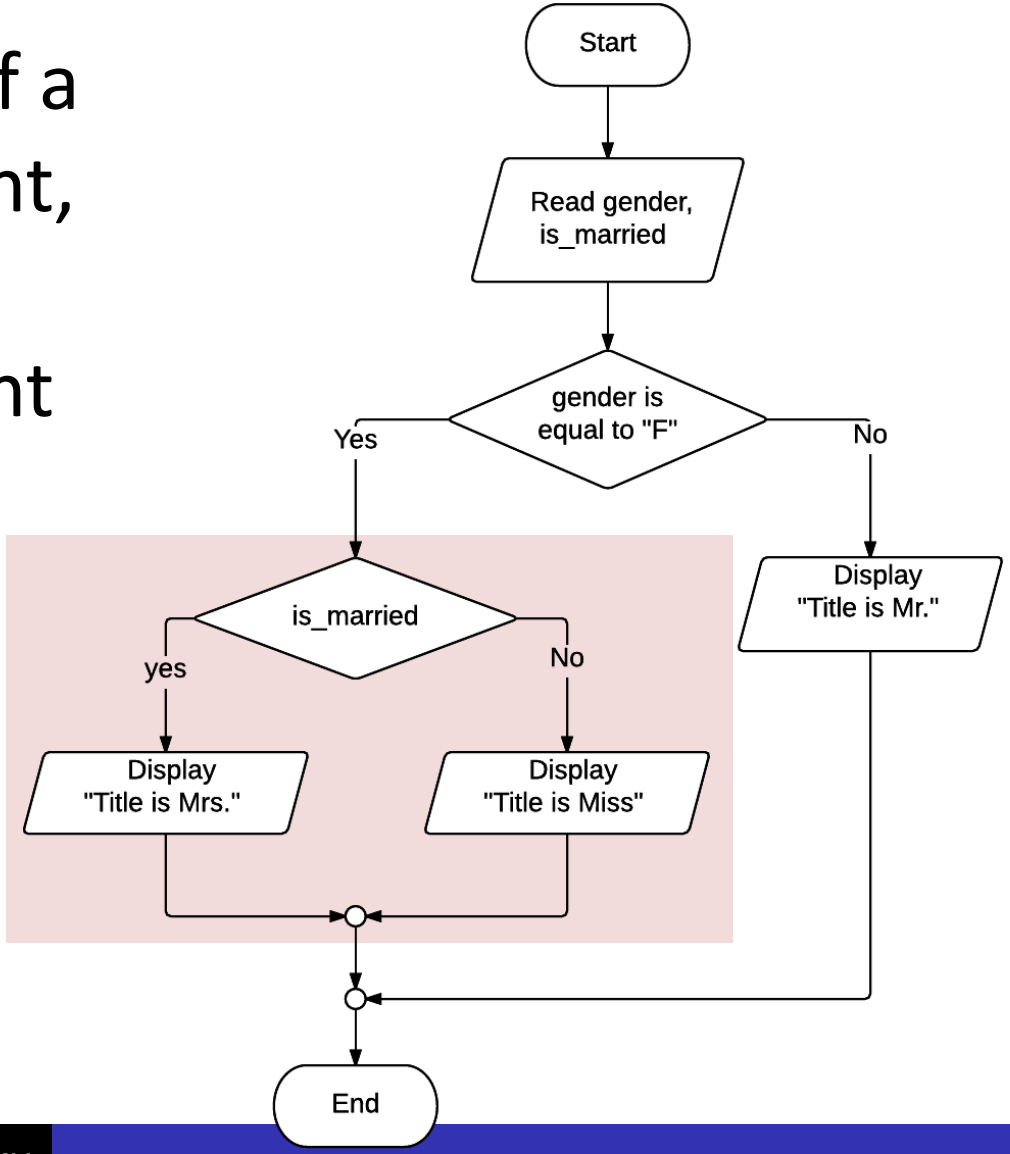
Grader Alert!
w06_1



Nested Conditionals

- Within any branch of a conditional statement, we can nest another conditional statement block.

```
if Boolean expression:
    if Boolean expression:
        block of code
    else:
        block of code
else:
    block of code
```



Nested Condition

- The program will monitor the car density of the traffic (measured in cars per kilometer, denoted as car_density) and the speed of the vehicle (measured in kilometers per hour, denoted as speed).
  - If the traffic is not heavily congested (car_density is less than or equal to 5 cars/km),
    - it is safe to drive the car at speeds up to 90 km/hr.
    - If the speed exceeds this limit, a warning should be issued.
  - However, if the traffic is congested (car_density is greater than 5 cars/km),
    - the car can only be driven at speeds up to 60 km/hr safely.
    - If the speed exceeds this limit, a warning should be given.

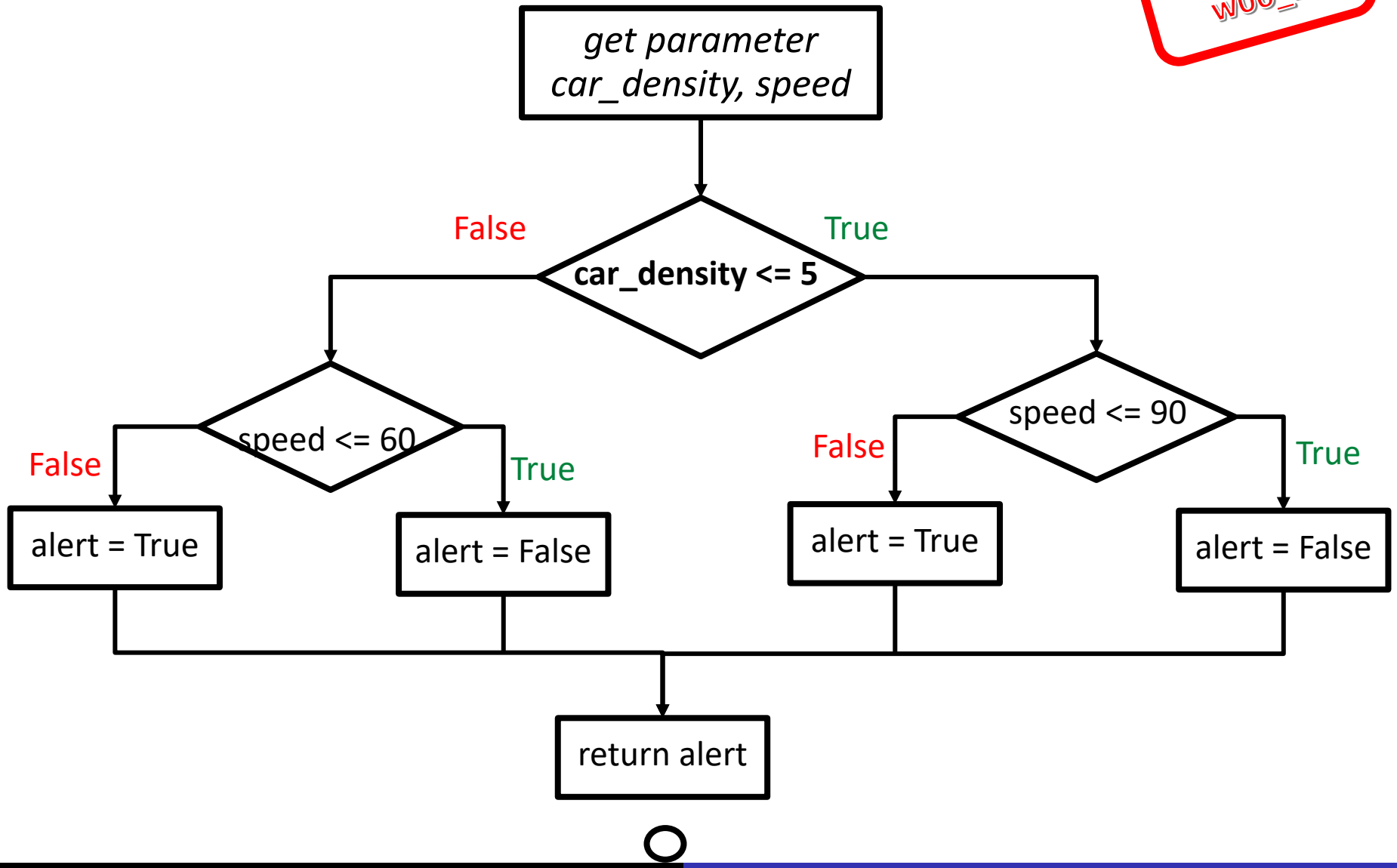
Break down the conditions

- traffic is not heavily congested (`car_density __ 5`)
  - speed ____90 km/hr. => Safe
  - otherwise => Warning

- traffic is congested (`car_density ___ 5`)
  - speed ____60 km/hr. => Safe
  - otherwise => Warning

speed_warning

Grader Alert!
w06_2



Leap Year

- A leap year is a year that has an extra day or month added to it in order to keep the calendar year aligned with the astronomical or seasonal year. To determine if a year is a leap year, the following rules can be applied:
  - If a year is not divisible by 4, it is considered a common year.
  - If a year is divisible by 4 but not by 100, it is considered a leap year.
  - If a year is divisible by both 100 and 400, it is considered a leap year.
  - For any other year that is divisible by 100 but not by 400, it is considered a common year.
  - By applying these rules, we can identify whether a given year is a leap year or not.

Example 1: Leap Year

- divisible by 4 but
  - ☑ not by 100 (2012, 2016 2020) — Case 1
  - ☑ divisible by both 100 and 400 (1600, 2000) — Case 3
  - ☒ divisible by 100 (1700, 1800, 1900) — Case 2
- STEP1: create test cases

	Test Case	output
Case 1	2012 2016 2020	YES
Case 2	1700 1800 1900	NO
Case 3	1600 2000 2400	YES
Case 4	2013 2014 2015	NO

Example 1: Leap Year

- STEP2: Consider case 1 and 4 first
  - divisible by 4 (2012, 2016 and 2020)

```
if year % 4 == 0:
    print("YES")
else:
    print("NO")
```

	Test Case	output
Case 1	2012 2016 2020	YES
Case 2	1700 1800 1900	NO
Case 3	1600 2000 2400	YES
Case 4	2013 2014 2015	NO

Example 1: Leap Year [2]

- STEP3: Consider Case 2
  - divisible by 4 but
    - ☒ divisible by 100 (1700, 1800, 1900)

```
if year % 4 == 0:
    print("YES")


else:
    print("NO")
```

	Test Case	output
Case 1	2012 2016 2020	YES
Case 2	1700 1800 1900	NO
Case 3	1600 2000 2400	YES
Case 4	2013 2014 2015	NO

Example 1: Leap Year [3]

- STEP3: Consider Case 2
  - divisible by 4 but
    - ☒ divisible by 100 (1700, 1800, 1900)

```
if year % 4 == 0:    //nested under case 1
    if year % 100 == 0:
        print("NO")
    else:
        print("YES")

else:
    print("NO")
```

	Test Case	output
Case 1	2012 2016 2020	YES
Case 2	1700 1800 1900	NO
Case 3	1600 2000 2400	YES
Case 4	2013 2014 2015	NO

Example 1: Leap Year [4]

- STEP3: Consider Case 3
  - divisible by 4 but
    - ☒ divisible by 100 (1700, 1800, 1900)
    - ☑ divisible by both 100 and 400 (1600, 2000)

```
if year % 4 == 0:
    if year % 100 == 0:
        print("NO")


    else:
        print("YES")

else:
    print("NO")
```

	Test Case	output
Case 1	2012 2016 2020	YES
Case 2	1700 1800 1900	NO
Case 3	1600 2000 2400	YES
Case 4	2013 2014 2015	NO

Example 1: Leap Year [5]

- STEP3: Consider Case 3
  - divisible by 4 but
    - ☒ divisible by 100 (1700, 1800, 1900)
    - ☑ divisible by both 100 and 400 (1600, 2000)

```
if year % 4 == 0:
    if year % 100 == 0:
        print("NO")
    elif year % 400 == 0:
        print("YES")
    else:
        print("YES")

else:
    print("NO")
```

	Test Case	output
Case 1	2012 2016 2020	YES
Case 2	1700 1800 1900	NO
Case 3	1600 2000 2400	YES
Case 4	2013 2014 2015	NO

Example 1: Leap Year [6]

- STEP5: Testing

```
if year % 4 == 0:
  if year % 100 == 0:
    print("NO")
  elif year % 400 == 0:
    print("YES")
  else:
    print("YES")

else:
  print("NO")
```

Solution: Swap the condition position.

	Test Case	output
Case 1	2012 2016 2020	YES
Case 2	1700 1800 1900	NO
Case 3	1600 2000 2400	YES
Case 4	2013 2014 2015	NO

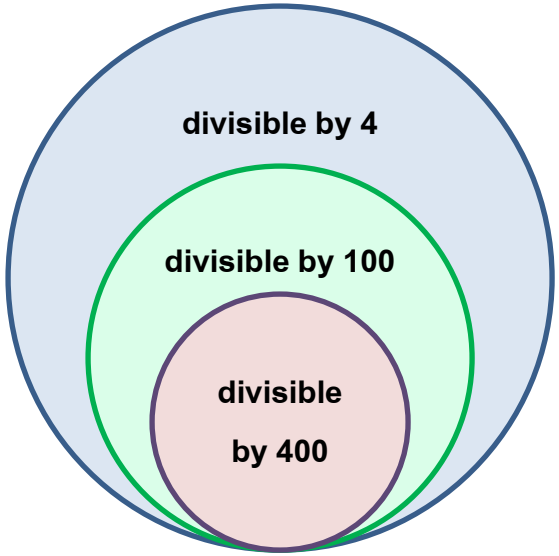
- Year = 2400 output = ?
- Why?
- How to fix the bug?

Example 1: Leap Year [7]

- STEP6: Reviewing

```
if (year % 4 == 0):
  if (year % 400 == 0):
    print("YES")
  elif (year % 100 == 0):
    print("NO")
  else:
    print("YES")

else:
  print("NO")
```

Where condition is a subset of each other (Not completely separated from each other.) Let's create a condition from a more specific case first. (small to large)



	Test Case	output
Case 1	2012 2016 2020	YES
Case 2	1700 1800 1900	NO
Case 3	1600 2000 2400	YES
Case 4	2013 2014 2015	NO

Lecture 2

Introduction to Python

Sequential Coding

Outline

1. Introduction
2. Variable
3. Assignment Statement
4. Data Type
5. Operators
6. Input and Output Statements
7. Sequential Coding
8. Lab Exercise

1.1 History of Python

- Developed in 1989 by Guido van Rossum while working for Centrum Wiskunde & Informatica (National Research Institute for Mathematics and Computer Science) in Amsterdam, Netherlands.
- Python was first released in 1991
- Python combines features from many languages (ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, etc.) with additional features such as easy-to-use interface.

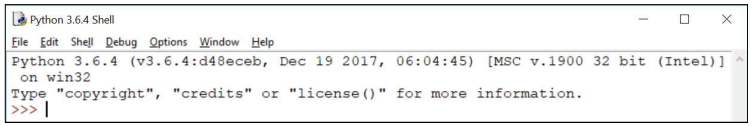
1.2 The Python Language

- Python is a high-level (3GL) language, with the following characteristics:

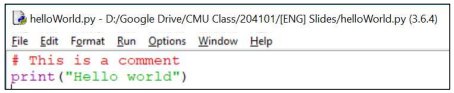
1. It use interpreter, translating and processing codes line-by-line, which allows:
2. Interactive mode, allowing users to run their code one commands at a time
3. It is currently, one of the most popular programming languages, especially for learning to program

Two Modes of Execution

- We will use IDLE, Python's default GUI (Graphical User Interface
- Interactive Mode
  - Work line-by-line, also serve as a main interface



- Script Mode
  - Run the whole file (a program)

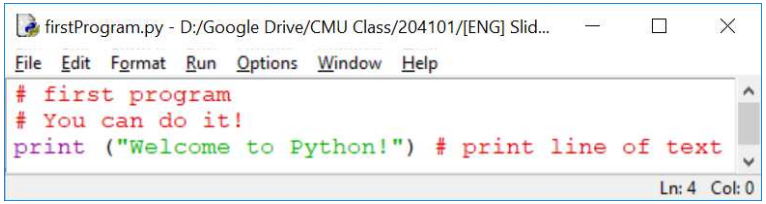


helloWorld.py

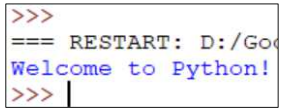
Run

Output (result) on interactive mode

1.3 The First Program



- Some Explanations
  - Line 1 is a comment (start with #) comments are just notes by/for programmers, and will not be run
  - Line 2 is another comment, # will denote that the rest of the text to the end of the line is comment
  - Line 3 contains print() command, which will print string (text) inside the parentheses.
- When you run (press F5) the program, you should get the following (blue text):



2. Variable

- Variables are used to reserve some memory to contain data. Variables can have different data types, and can require different memory sizes.
- In Python, there are some rules in naming variables.

1. It must begin with a letter (a - z, A - B) or underscore (_)
2. Other characters can be letters, numbers (0-9) or _
3. Cannot be reserved words
4. No space allowed in variable's name
5. Can be any (reasonable) length
6. The name is case-sensitive, capital letters and lowercase letters are considered different characters.

   Example: NUM, Num, and num are considered different variables

Reserved Words

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Examples of Variable Naming

Variable Name	Valid?	Why Not?
123MyID	✘	Cannot start with number
_ThinkBig	✓	
mylife@CMU	✘	Cannot contain @
Pin number	✘	Cannot contain space
Dorm_number	✓	

3. Assignment Statement

- Assignment statement is how we give value to a variable
- In other languages, variables need to be declared before usage. But in python, variables are automatically declared the first time they are assigned values
- We assign value to a variable using = operator, for example:

```
score = 75
# score is an integer variable with the value of 75
gpa = 2.85
# gpa is a floating point (real number) variable with the value of 2.85
```

NOTE: In Python, you can actually assign values to multiple variables in one line, for example:

```
score, gpa = 75, 2.85
```

Will do the same as the statements above.

Examples of Assignments

- You can also do chain assignment, where all the variables in the chain will be assigned the value of the rightmost variables/constant

Example 1

```
a = b = c = 1
# a, b, and c will be integers, with the value of 1
```

Example 2

```
a, b, c = 1, 2 , “john”
# a and b will be integer, with values of 1 and 2 respectively
# c will be string, containing john
```


4. Data Type

- The followings are basic data types you should know:

1. int — contains integer number
2. float — contains real number
3. boolean — contains True/False value
4. string — contains text
5. list — contains multiple items

4.1 Number Data Type (int, float)

- int contains integer, which are number without decimal point. Example: 2, -50, 1009
- float contains real number, sometime called floating point number. Real number can have decimal point. Example: 15.20, -21.9

4.2 Boolean Data Type

- Boolean variable can either be True, or False, usually a result of a comparison. For examples:
  - 4 > 1 will get you True
  - 6 < 7 will get you False
- boolean can be very useful in programming. It is use in selection (if statement) to have program does different things, based on comparison results.

4.3 String Data Type

- string is used to contain text. In python string can be denote by quotation marks, and can use single ('), double (") or triple (''' or """) marks.
- Examples:

```
word = 'word'
sentence = "This is a sentence."
```

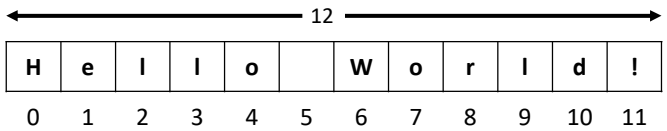
- Triple marks are use for string longer than a single line

```
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

- Furthermore, you can use +, * operators
  - + will concatenate two or more strings together
  - * will repeatedly concatenate one string multiple times

4.3 String Data Type (cont.)

- You can also access part of string, which we will call substring, by using [] or [:] and index (location)
- Characters in a string are stored by index, from 0 to end – 1
- “Hello World!” example:



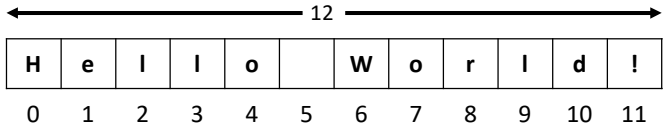
  - The string has the length of 12
  - The first character (index 0) is ‘H’
  - The last character (index 11) is “!”

Examples: Working with String

```
str = 'Hello World!'
print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th (end - 1 is 5-1 = 4)
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST ")# Prints concatenated string
```

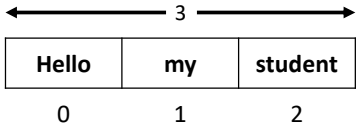
This will produce the following result:

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```



4.4 List

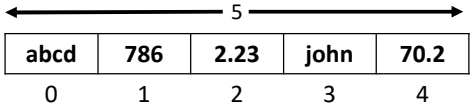
- list is a container data type. A list can have multiple items.
- Items in the list can be of multiple data types (compound data type)
- When displayed, items are shown inside [ ] and are separated by comma (,)
- We can get values of item using [ ] and [:] and index values
- Like string, the first item is at index 0, and the last item is at end – 1
- Example: myList = [‘Hello’, ‘my’, ‘student’] will looks like this



Examples: Working with List

```
exlist1 = ['abcd', 786 , 2.23, 'john', 70.2 ]
exlist2 = ['Hello' ,'my' , 'student']

print (exlist1)          # Prints complete list
print (exlist1[0])       # Prints first element of the list
print (exlist1[1:3])     # Prints elements starting from 2nd till 3rd (end - 1 is 3 – 1 = 2)
print (exlist1[2:])      # Prints elements starting from 3rd element
print (exlist2 * 2)      # Prints list two times
print (exlist1 + exlist2)# Prints concatenated lists
```

This will produce the following result:




```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
['Hello', 'my', 'student', 'Hello', 'my', 'student']
['abcd', 786, 2.23, 'john', 70.2, 'Hello', 'my', 'student']
```


5. Operators

- Operators are symbols we used to perform some action/operations on operands (variables or constants). Today, we will look at

1. Arithmetic operators
2. Comparison (relational) operators
3. Assignment operators
4. Logical operators
5. Bitwise operators
6. Membership operators
7. Identity operators

5.1 Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example	Result
+	Addition	a + b	30
-	Subtraction	a - b	-10
*	Multiplication	a * b	200
/	Division	b / a	2
%	Modulus	b % a	0
**	Exponent	2**3	8
/ /	Floor Division	9//2	4
		9.0//2.0	4.0

5.2 Comparison Operators

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example	result
==	equal	a == b	False
!=	not equal	a != b	True
>	greater than	a > b	False
<	less than	a < b	True
>=	greater than or equal	a >= b	False
<=	less than or equal	a <= b	True

5.3 Assignment Operators

- Beside straight assignment (=), you can also use:

Operator	Example	Explanation
=	c = a + b	
+=	c += a	c = c + a
-=	c -= a	c = c - a
*=	c *= a	c = c * a
/=	c /= a	c = c / a
%=	c %= a	c = c % a
**=	c **= a	c = c ** a
//=	c //= a	c = c // a

5.4 Logical Operators

Assume variable c holds True and variable d holds False, then:

Operator	Example	Result
and	c and c	True
	c and d	False
or	a or b	True
not	not(a or b)	False

Truth Table

P	Q	P and Q	P or Q	P	Not P
True	True	True	True	True	False
True	False	False	True	False	True
False	True	False	True		
False	False	False	False		

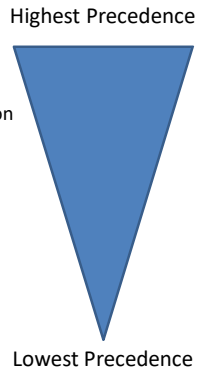
5.5 Other Operators

- There are other operators that will not be covered (yet)
  - Bitwise operators work on number bit by bit: >>, <<, &, |
  - Python Membership Operators: in , not in
  - Python Identity Operators: is , is not

5.6 Operators Precedence

- To decide which operator will be performed first, Python follow the precedence denote in the table below:
- For operators with the same precedence, perform the left one first

Operator	Description
()	parentheses
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators



Translating Mathematical Expression into Python

- You might be familiar with a certain style of writing an expression, but it might not be understood by Python, and you need to convert them.
- Examples:

Mathematical Expression	Python Expression
xy - 5z	x * y – 9 * z
x2 + 4y + 5	x**2 + 4 * y + 5
8y2 - 8z	8 * y**2 - 8 * z

Example: Operators Precedence #1

Find the result of c * d – x / y, with the following assignments

x = 16

y = 4

c = 2.5

d = 0.25

Solution

```
c * d – x / y
2.5 * 0.25 – 16 / 4  # assignment
0.625 – 16 / 4       # perform *
0.625 - 4            # perform /
-3.375               # perform –
```

The answer is -3.375

Example: Operators Precedence #2

- Find the result of -(-5) * (x % y – x // (y+1)), with the following assignments

  x = 16

  y = 4

  c = 2.5

  d = 0.25

Solution

-(-5) * (x % y – x // (y+1))
-(-5) * (16 % 4 – 16 // (4+1))
-(-5) * (16 % 4 – 16 // (5))
5 * (16 % 4 – 16 // 5)
5 * (0 – 3)
5 * (– 3)
-15

The answer is -15

Example: Operators Precedence #3

- Find the result of c // d + y % x != 2*c – d, with the following assignments

  x = 16

  y = 4

  c = 2.5

  d = 0.25

Solution

c // d + y % x != 2*c – d
2.5 // 0.25 + 4 % 16 != 2*2.5 – 0.25
10 + 4 != 5 – 0.25
14 != 4.75
True

The answer is True

Some Useful Functions in Python

- int(A)
  - Convert argument (Value of variable A, in this case) into integer
  - Also need a variable to assign the integer to
  - Try A = int(3.142) ⟶ What is the value of A?
- float(A)
  - Similar to int(), but will convert the argument into floating point number instead

Some Useful Functions in Python (cont.)

- Note: you can have nested function call (one function inside another). The innermost function will be called first, and the result will be argument for the one right outside it
- For example: A = int(input("Enter a number"))
  - input("Enter a number") will be called first, prompting the user to enter the number
  - The result of input(), which is a string, will then be converted into an integer by int(), and then assigned to variable A

6. Input-output Statements (Recap)

- For a program to be able to interact with the user, the program need a way to:
  - Receive data from the user (input), and
  - Display result to user (output)
- We will start with:
  - Output: print() function
  - Input: input() function

6.1 print()

- print() will display its arguments (values inside the parentheses) on interactive mode windows.
- You can print multiple strings/variables at the same time by separating them with comma (,). The texts will be concatenated when printed.
- If you want the text to start a new line, you can add new line character (\n) in the text
- You can also use \t for tab

6.2 input()

- input() function will prompt user (with the message inside the parentheses) to enter data. User will type out data, then press enter.
- Received input will be of string type, and we need a variable to hold the data (function return)

Example:

```
str = input("Enter your input: ")
print ("Received input is : ", str)
```

Result (user input in blue):

```
Enter your input: Hello Python
Received input is : Hello Python
```

- From the example, “Hello Python” will be stored in variable str

6.2 input() (cont.)

- Since the input will be string (text), it cannot be used in calculation right away
- We will need to convert the input into appropriate type first (int or float)

Example:

```
inp1 = input("Input  integer number  : ")    # prompt user for input
                                             # store the result on inp1
no1 = int(inp1)                              # convert result in inp1 to int
                                             # store it on no1
inp2 = input("Input  float number  : ")      # get next user input into inp2
no2 = float(inp2)                            # convert to float, store on no2
```

6.2 input() (cont.)

- Warning: be careful about input data type.

Example:

```
inp=input("Input  number  : ")
no=int(inp)
```

```
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    no = int(inp)
ValueError: invalid literal for int() with base 10: '1.2'
```

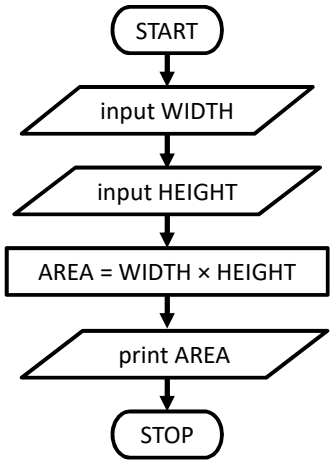
- If the user input 1.2, it will read and put into inp with problem. BUT when Python try to convert it to integer, an error will occur (since 1.2 is not an integer)

7. Writing Sequential Code

- Translate from design (description, pseudocode, or flowchart) into programming code
- You might need to do program analysis and program design first!
- Start from the top, moving downward and end at the bottom

7.1 Coding Example #01

- Finding rectangle's area




```
#Receive length, width from input
LENGTH = float(input("Input length : "))
WIDTH = float(input("Input width : "))

#Calculate area
#using formula "area = length * width"
AREA = LENGTH * WIDTH

#Display the result
print("Area =", AREA)
```


7.2 Coding Example #02

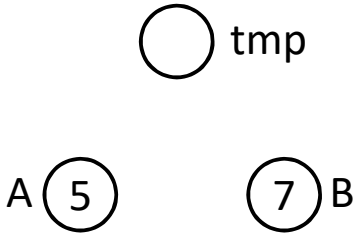
- Swap the values of two variables

"Take two already-assigned variables, A and B. Swap the values between them, then print the result"

```
#Start with assignment
A = 5
B = 7
print("A =", A)
print("B =", B)

#Put code for swapping A and B here

#End swapping

#Show the results
print("A =", A)
print("B =", B)
```

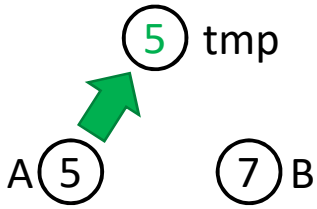
7.2 Coding Example #02 – Swapping Concept

- VERY IMPORTANT: A code will be executed one line at a time, so If you do A = B first, the original value of A will be lost
- You will need temporary variable (tmp)

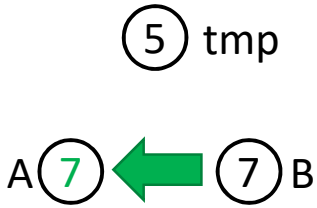


7.2 Coding Example #02 – Step by Step

1. tmp = A # assign the value of A to tmp

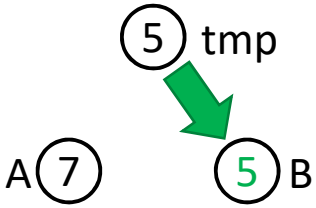


2. A = B # assign the value of B to A



7.2 Coding Example #02 – Step by Step (cont.)

- B = tmp # assign value of tmp (A's original values) to B



- And we have A = 7 and B = 5!

7.2 Coding Example #02 – Complete Codes

```
#Start with assignment
A = 5
B = 7
print("A =", A)
print("B =", B)

#Put code for swapping A and B here
temp = A  # use temp as temporary to hold A's value
A = B     # assign B's values to A
B = temp  # assign temp's (A's origignal) values to B
#End swapping

#Show the results
print("A =", A)
print("B =", B)
```

7.3 Coding Example #03

- Finding Mean Average of Three Numbers

"Receive three numbers from the input, the display the mean of those three numbers"

- Note: use float() for the inputs
- The program should work like this:

```
Input first number  : 14.32
Input second number : 15.44
Input third number  : 17.29
Mean = 15.683333333333332
>>>
```


7.3 Coding Example #03 – General Idea

- So first, some questions:
  - What is(are) the input(s) of this program?
  - Output(s)?
  - Process? Or, how do we calculate the mean of three numbers?
- Can you then describe the algorithm?
  - What are the steps you need to do to solve the problem, starting from: where are you going to get the input data from?
- If you're not sure about the codes yet, write the steps down in comments first, then do the coding for each step

Example #03 – Complete Codes

```
# received three number from input
num_01 = float(input("Input first number  : "))
num_02 = float(input("Input second number : "))
num_03 = float(input("Input third number  : "))

# calculate the mean, which the sum of the numbers
# divided by the count (3)
meanNum = (num_01 + num_02 + num_03)/3

# show the mean
print("Mean =", meanNum)
```


Practice

- Find BMI
- Find an average of three numbers
- Find the circumference of a circle
- Find the area of a circle

Reference

- Mark Lutz, "Programming Python", O'Reilly, 1996.
- Deitel ,"Python How to program", Prentice-Hall,Inc., 2002.
- Matt Telles , "Python Power !", Thomson Course Teachnology,2008.
- python 3 help documentation
- PYTHON TUTORIAL Simply Easy Learning by tutorialspoint.com

3. **(w05_3_xxxxxxxxx.py) find_max [Attachment]** Create a function named `find_max` that takes 2 parameters: `num1` and `num2`. The function must evaluate and **return the highest number** between 2 parameters. The main program will print out the result on screen

   Function Specification:

```
def find_max(num1,num2):
    # Return max value between num1 and num2
```

Input	**Output**
3 5	5
75 2	75
33 -73	33
22 22	22

4. **(w03_4_xxxxxxxxx.py) pass_all_subjects [Attachment]** Create a function name pass_all_subjects that takes 3 parameters: math,sci, eng. Which are the score of each subject. The function much evaluate and return if the student has passed all subjects, which requires a minimum score of 50 in each subject. Return True if the student pass all subject and return False if they not pass all subject

Hint: you can create another function to check for pass criteria and use logical operator to combine the conditions

Function Specification:

```
def pass_all_subjects(math,sci,eng):
    #perform the comparison between each pair of number in parameters
    #return True if the student pass all subject
    #return False if they not pass all subject
```

Input	**Output**
85 72 53	True
54 85 12	False
48 66 35	False
45 35 48	False

5. **(w03_5_xxxxxxxxx.py) max_of_three [Attachment]** Create a function named `max_of_three` that takes 3 parameter: `num1,num2,num3`. The function must evaluate and **return the maximum number** among parameters

Hint: You can reuse the function find_max() by copy the function definition from w05_3 and place it in the same file

Function Specification:

```
def max_of_three (num1,num2,num3):
    #perform the comparison between each pair of number in parameters
    #return the maximum value among them
```

Input	**Output**
2 5 8	8
7 11 2	11
-3 -5 -2	-2
5 0 -2	5

Submission

1. The format and order of input/output messages must follow the example provided during the program execution.
2. The submitted program file must include comments at the beginning of the file as specified in the course Canvas instructions.
3. The submitted program file must include pseudocode comments for each step of the program as specified.
4. Upload the source code file to the designated homework submission website specified for each task at http://cmu.to/grader204101
5. Check the grader instructions from https://cmu.to/instruction

2. **(w06_2_xxxxxxxxx.py) speed_warning[Attachment]:** Create a function named `speed_warning` that takes 2 parameters: **`density and speed`**. The function to issue warnings for a car based on the current traffic conditions. The program will monitor the car density of the traffic (measured in cars per kilometer, denoted as carDensity) and the speed of the vehicle (measured in kilometers per hour, denoted as speed).
    a. If the traffic is not heavily congested (density is less than or equal to 5 cars/km), it is safe to drive the car at speeds up to 90 km/hr. If the speed exceeds this limit, a warning should be issued.
    b. However, if the traffic is congested (density is greater than 5 cars/km), the car can only be driven at speeds up to 60 km/hr safely. If the speed exceeds this limit, a warning should be given.
    c. The function should return a Boolean value according to warning status, True for the situation that a warning should be issued and False for the safety situation.

NOTE: delete "pass" keyword before edit the function in template file

Input	**Output**
`car_density = 4` `car_speed = 80`	`False`
`car_density = 7` `car_speed = 70`	`True`
`car_density = 4` `car_speed = 100`	`True`
`car_density = 7` `car_speed = 60`	`False`

3. **(w06_3_xxxxxxxxx.py) bmi_translate [Attachment]** Create a function called bmi_translation that takes the calculated BMI (Body Mass Index) as input and returns the corresponding translation based on the BMI range. The function should have the following specifications:
   Function Name: bmi_translation
   Parameters: bmi (float) - The calculated BMI value
   Return: translation (string) - The translation of the BMI based on the range

   •Translate the BMI into the following categories:
   - If the BMI is less than 18.5, return "Underweight".
   - If the BMI is between 18.5 and 24.9 (inclusive), return "Normal weight".
   - If the BMI is between 25.0 and 29.9 (inclusive), return "Overweight".
   - If the BMI is 30.0 or greater, return "Obese".

   NOTE: delete "pass" keyword before edit the function in template file

Input	**Output**
`Input weight in kilograms: 55` `Input height in centimeters: 150`	`BMI: 24.444444444444443` `Transation of BMI: Normal weight`
`Input weight in kilograms: 45` `Input height in centimeters: 156`	`BMI: 18.49112426035503` `Transation of BMI: Underweight`
`Input weight in kilograms: 80` `Input height in centimeters: 179`	`BMI: 24.968009737523797` `Transation of BMI: Obese`

4. **(w03_4_xxxxxxxxx.py) days_in_month [Attachment]** function called `days_in_month` that takes 2 integer parameters: `month` and `year`. The function should evaluate and returns an integer representing the number of days in that specific month and year.
   - February (month number 2):
     - If the given year is a leap year (determined using the is_leap_year function), February has 29 days.
     - If the given year is not a leap year, February has 28 days.
   - April, June, September, and November (month number 4, 6, 9, 11):
     - These months always have 30 days.
   - All other months (January, March, May, July, August, October, December):
     - These months have 31 days.

The function should consider leap years using the is_leap_year function.

NOTE: delete "pass" keyword before edit the function in template file

Input	**Output**
`month = 2` `year = 2020`	`29`
`month = 4` `year = 2021`	`30`
`month = 12` `year = 2022`	`31`

5. **(w06_5_xxxxxxxxx.py) Zodiac Sign Determination [Attachment]** Write a function called zodiac that takes two parameters: day and month, representing a person's birthdate. The function should determine the corresponding zodiac sign based on the astrological zodiac calendar and return it as a string.

   •The function should follow these specifications:
   - Validate the input to ensure the day value is within the range of 1 to 31 and the month value is within the range of 1 to 12
   - Determine the zodiac sign based on the provided birthdate using if-elif-else statements.
   - Match the birthdate with the corresponding zodiac sign based on the following table:
     - January 20 - February 18: Aquarius
     - February 19 - March 20: Pisces
     - March 21 - April 19: Aries
     - April 20 - May 20: Taurus
     - May 21 - June 20: Gemini
     - June 21 - July 22: Cancer
     - July 23 - August 22: Leo
     - August 23 - September 22: Virgo
     - September 23 - October 22: Libra
     - October 23 - November 21: Scorpio
     - November 22 - December 21: Sagittarius
     - December 22 - January 19: Capricorn
   - If the day and month values do not correspond to a valid birthdate, return the string "Invalid date".
   - **NOTE: delete "pass" keyword before edit the function in template file**

Input	**Output**
`day = 29` `month = 2`	`Invalid date`
`day = 18` `month = 6`	`Gemini`
`day = 23` `month = 12`	`Capricorn`
`day = 10` `month = 9`	`Virgo`

Week 06 Condition II

Specification
In the requirement that states [Attachment], please download the Template file from the Grader and submit only the file(s) that have the specified names for each respective task.

1. **(w06_1_xxxxxxxxx.py) ticket_drawing[Attachment]:** Write a function named `ticket_drawing` that takes 1 parameter: **`num`** and determines the prize and return prize in string (as shown in example) based on the following conditions:
   - If the number is between 45 and 55 (inclusive), the function should return "First prize".
   - If the number is between 15 and 30 (inclusive) or between 75 and 90 (inclusive), the function should return "Second prize".
   - For any other number, the function should return "No prize".
   - The input number will always be within the range of 1 to 99.

   NOTE: delete "pass" keyword before edit the function in template file

Input	**Output**
`50`	`First prize`
`25`	`Second prize`
`70`	`No prize`
`90`	`Second prize`