Today's Topics:

- Designing programs
- Using Math module
- Functions and input

Lee wilkins

Office : 6C.9

Contact: MIO

INTRODUCTION TO COMPUTER PROGRAMMING IN ENGINEERING AND SCIENCE

Comp Sci assignments (All tests and assignments occur Wednesdays in our lab block)

Assignment 1: week 4 (10%)

Test 1 week 7 (15%)

Assignment 2 week 8 (10%)

Assignment 3 week 11 (10%)

Test 2 week 13 (15%)

Physics assignments

Assignments (4 x 2%) 8% Date communicated by the Physics teacher

Project 1: Solving differential equations 10% Week 11

Project 2: Applying programming in science 22% Week 15

GRADE BREAKDOWN REVIEW

Wednesday: non graded practice quiz on Moodle!

GRADE BREAKDOWN REVIEW

There are various stages to develop a program:

1. Problem definition
2. Program design
3. Program coding
4. Program debugging
5. Program testing
6. Program maintenance

PROGRAM PLANNING

Understanding a problem is half the answer!

Understand and identify the problem for which the software is to be developed.

It helps to write it out in a clear sentence, for example.

Program purpose: Why does this program exist?

Input: What do we need to give this function to work?

Output: What is the desired end result?

PROGRAM DEFINITION

Software developer use tools such as pseudo-code, algorithms and flowcharts to design a solution of the problem

Pseudo code: A human-readable analogous to code that helps you think through how your program is structured.

Algorithms: Clear instructions on how to perform a task

Flow chart: Visual diagram of how your program works

PROGRAM DESIGN

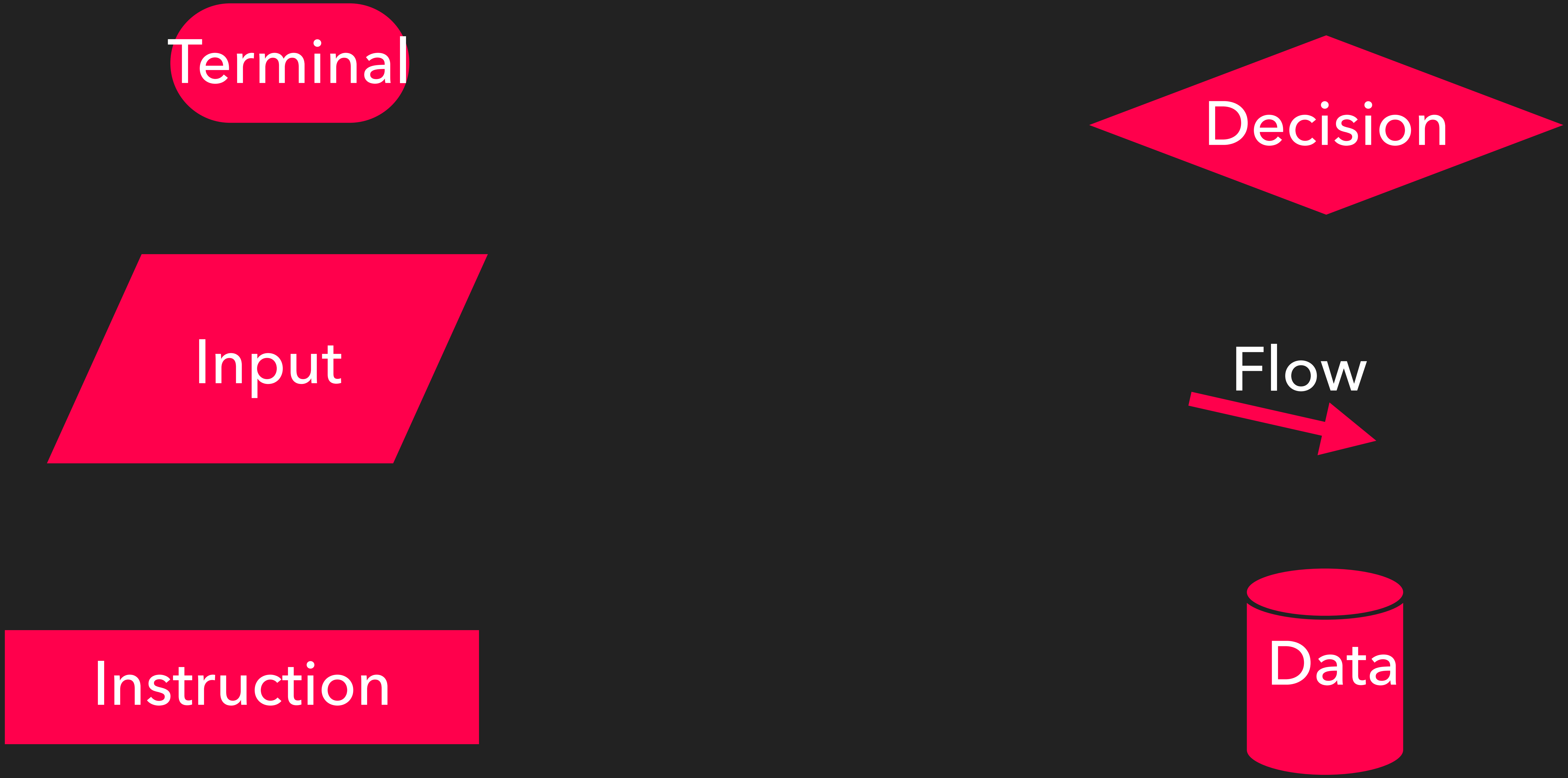
Pseudo code is a way of talking through your program using programming-like words. Talking out loud as well as writing it out can help you understand your program a lot better. Example of pseudo code:

Program Purpose: Determine if the day is good or bad.

Input: cloud status

Output: Weather is good or bad

PSEUDO CODE



Terminal

Decision

Input

Flow

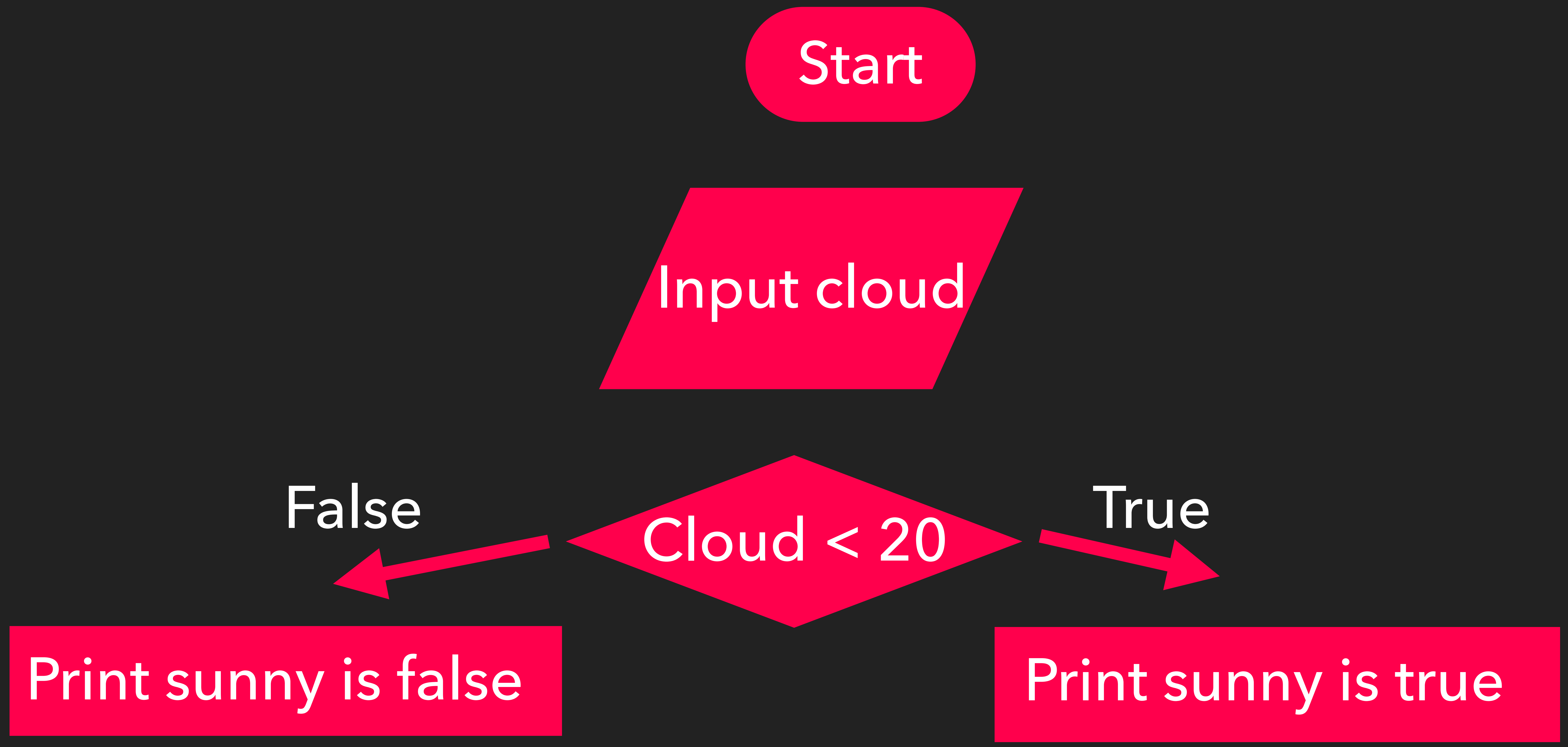
Instruction

Data

FLOW CHARTS

```
Ask the user to input cloud status.

If cloud status is less than 20, then sunny is true, else, sunny is false.

If sunny is true,  the weather is good.

If sunny is false, the weather is bad.

Print the weather
```

PSEUDO CODE


Start
Input cloud
Cloud < 20
False
True
Print sunny is false
Print sunny is true

FLOW CHARTS

Now we can begin to actually code your program. Program design is translated into a formal computer program. Reference your design process when writing!

PROGRAMMING

Logical errors are detected and rectified.

Some logical errors

- a loop that is infinite (program hangs)
- wrong expression due to operators precedence rules
- an if statement that always evaluates to True
- using assignment instead of equality

TESTING / DEBUGGING

5. Program testing

Use test cases to test the program

- Test if requirements are met
- Identify and test most special cases (extreme)

For e.g., max and min values of all variables as test data.

Try out your program by putting in strange data or variables and seeing how it reacts.

TESTING

Program needs maintenance when:

- new requirements are proposed
- some bugs are actually identified by customer

Its always good to consider how maintenance might work when writing your code. Ex: can a function be made more general? Is it easy to expand on what you've written or will adding new features require a complete rewrite?

MAINTENANCE

MATH MODULE

```
import math
```

This gives access to all kinds of things to help you do more advanced calculations. You can see all of them here.

You need to put this at the top of your document or above all of the math module methods you use.

https://www.w3schools.com/python/module_math.asp

```python
print(math.pi) # pi
print(math.e) # Euler’s number
print(math.tau) # tau
```

By importing math we have access to a number of different methods and constants. These are constants that don’t have () because they are not performing tasks, they are representing numbers.

CONSTANTS

```python
my_num = math.sqrt(4)
print(my_num)
```

You can pass numbers through a method as a parameter (or param). In this example, we are passing 4 through the square root function (math.sqrt())

You can alway look up how it works

https://www.w3schools.com/python/ref_math_sqrt.asp

METHODS

You can convert degrees and radians by passing them THROUGH the appropriate methods. You can pas a method a parameter by putting it in the round brackets like this:

```
print(math.degrees(1))
# this takes radians, prints degrees
print(math.radians(90))
# this takes degrees, prints radians
```

You can also pass variables through a function

```
my_angle_degrees = 90
my_angle_radians = math.radians(my_angle_degrees)
print(f'{my_angle_degrees} degrees is {my_angle_radians} radians’)

another_angle_radians = 1.1
another_angle_degrees = math.degrees(another_angle_radians)
print(f'{another_angle_degrees} degrees is {another_angle_radians} radians')
```

USING MATH.

```
my_num = math.pow(2,2)
print(my_num)
```

Some methods need multiple parameters, such as pow. This needs to know the base and the exponent

https://www.w3schools.com/python/ref_math_pow.asp

USING MATH.

```
You can pass methods through other methods. Like this:

myNum = 2.0

print(math.sqrt(math.pow(myNum, 2)))
> 2.0

print(math.log(math.exp(myNum)))
> 2.0

print(math.log10(math.pow(10, myNum)))
> 2.0
```

USING MATH.

```
opposite_side = 5
adjacent_side = 3

These two lines are the same
hypotenuse_side = opposite_side**2 + adjacent_side**2
This is the same line using math.pow
hypotenuse_side = math.pow(opposite_side, 2) + math.pow(adjacent_side, 2)


I can apply the square root function after
hypotenuse_side = math.sqrt(hypotenuse_side)


You can also do it all in one expression like this
hypotenuse_side = math.sqrt(math.pow(opposite_side, 2) + math.pow(adjacent_side,
2))
print(f'The triangle hypotenuse is {hypotenuse_side}’)

Remember, it is not faster to do all the calculations in one line. Don’t hesitate
to spread it out and use multiple lines. Make it easy to read!
```

USING MATH.

Calculate the adjacent angle with the two sides, your result is in radians.

```
angle_adjacent = math.atan(opposite_side/adjacent_side) Returns the value in radians
```

Convert it to degrees like this

```
angle_adjacent = math.degrees(angle_adjacent)
```

Or all in one expression where degrees holds the entire expression

```
angle_adjacent = math.degrees(math.atan(opposite_side/adjacent_side))
```

USING MATH.

A series of steps that we've already told the computer how to do

When we need the computer to make the same "recipe" multiple times, we can simply call the function instead of telling it all the steps again.

FUNCTIONS

Functions can have **arguments** or **parameters**)

Arguments can be required or optional

The number of arguments depends on the function

Some functions may also require arguments to be a certain type

Functions can also give us something they end (**return** a value)

We've already used some built-in functions like print() and type(), but now we're going to write our own

FUNCTIONS

def means we are defining the function

This is your function name

Note the syntax, you always need () so you know its a function

```
def my_function():
    print("this is my function")
```

Function content is indented

Your function ends, here, but it wont be called until you do this:

```
my_function()
```

This is where your function actually runs. This is called calling the function.

FUNCTION SYNTAX

PASSING PARAMETERS

I can pass a parameter through my function. It will be used by this name only within the scope of the function.

Whatever I pass through the function will be placed here

```
def my_function_param(my_string):
    print(my_string)

my_function_param("this is a string")
my_function_param("I can print anything here")
my_function_param("This function is so useful!")
```

We can call the same function to produce different results

You can create variables inside your function. They only exist inside the function and not int he rest of your program

You can pas multiple parameters through your function

```
def calculate_square_area(width, height):
    area = width * height
    print(f'The area of a square with the width of {width} inches ad the height of {height} inches is {area} inches^2')

calculate_square_area(2,4)
# The area of a square with the width of 2 inches ad the height of 4 inches is 8 inches^2
calculate_square_area(5,10)
# The area of a square with the width of 5 inches ad the height of 10 inches is 50 inches^2
calculate_square_area(100,3)
# The area of a square with the width of 100 inches ad the height of 3 inches is 300 inches^2
```

MULTIPLE PARAMETERS

The function ends when
the indentation ends

```
def calculate_square_area(width, height):
    area = width * height
    print(f'The area of a square with the width of {width} inches ad the height of
{height} inches is {area} inches^2')

print(area)

# NameError: name 'area' is not defined
```

Area only exists inside the function. This idea is called scope

SCOPE

```
def return_calculate_square_area(width, height):
    area = width * height
    return(area)

print(return_calculate_square_area(2,4))

my_area = return_calculate_square_area(2,4)
```

The word return is used to give back a value

In this example we are returning the area, either printing it or storing it

RETURN

INPUT

This function takes no params

```
def my_input_function():
    take_my_number = int(input("What is the number? "))
    what_to_add = int(input("What do I add to it? "))
    print(take_my_number + what_to_add)

my_input_function()
```

Inputs are requested
inside the function

Input taken outside the function

```
outside_function_input = input("Pass this: ")

def outside_input_function_one(my_string):
    my_inside_string = " What did we pass through this function? "
    print(f'{my_inside_string} {my_string}')

def outside_input_function_two(my_string):
    my_inside_string = " I can use this in two different ways "
    print(f'{my_inside_string} {my_string}')

outside_input_function_one(outside_function_input)
outside_input_function_two(outside_function_input)
```

Inputs are passed through the function

INPUT

A NOTE ON FUNCTION NAMES...

Current best practice is to use snake_case

However, you will also see camelCase and flatcase sometimes

e.g. analyzeData(), endswith()

Python doesn't care, but humans do!

Humans who might read your code: your future coworkers, your future self, your teacher who will be grading you (me!)

Get into the habit of following best practices

Most importantly: be consistent!

LAB TIME

Coming up: Functions, nested if statements, coding standards

Next Class: Ungraded quiz in class

Next week (wednesdnday) in class assignment (graded)

NEXT CLASS: