KETE HS24

# Introduction to Python Programming

TUTORIAL 1

repl.it

Created by Daniel Benninger

# Inhaltsverzeichnis

# Introduction to Python Programming

## Coding in General

repl.it

Created by Daniel Benninger

# What is Programming?

- **Programming** is the process of creating a set of instructions that tell a  computer how to perform a task.

- Programming can be done using a  variety of computer "languages," such as SQL, Java, Python, and C++.

*https://www.khanacademy.org/computing/computer-_programming/programming/intro-to-programming/v/programming-_intro*

repl.it

# What is Programming?

- **Coding** is the method of giving instructions to a computer to perform  a specific task.

- You may have also heard it referred to as “software programming” or “computer programming.”
- These instructions are  communicated using a “computer language” that computers can  understand. These languages include visual blocks, Java, Python, and C, C++

repl.it

# What is Programming?

- **What is Coding?**

## Communication

Coding is a language used to communicate instructions to a computer.

## Problem Solving

Coding involves breaking down large problems into smaller, manageable steps.

## Creation

Coding enables the creation of software, websites, games, apps, and much more.

# What is Programming?

- **Learning to code** expands problem solving and critical thinking  skills, making it a great opportunity for younger people to build those skills while young

repl.it

# What is Python?

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar
- High-level internal object data types
- Automatic memory management
- It’s free (open source)!

repl.it

# What is Python?

Python is an **interpretive language.**

- This means that your code is not directly run by the hardware.
- It is instead passed to a *virtual machine*, which is just another programme that reads and interprets your code.
- If your code used the '+' operation, this would be recognised by the interpreter at run time, which would then call its own internal function 'add(a,b)', which would then execute the machine code 'ADD'.
- This is in contrast to compiled languages, where your code is translated into native machine instructions, which are then directly executed by the hardware. Here, the '+' in your code would be translated directly in the 'ADD' machine code.

repl.it

# What is Python?

Some Language Properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading
- Indentation for block structure

# Why Learn Python?

- Fun-to-use "Scripting language"
- Object-oriented
  - Highly educational
- Very easy to learn
- Powerful, scalable, easy to maintain
  - High productivity
  - Lots of libraries

# Where to Use Python?

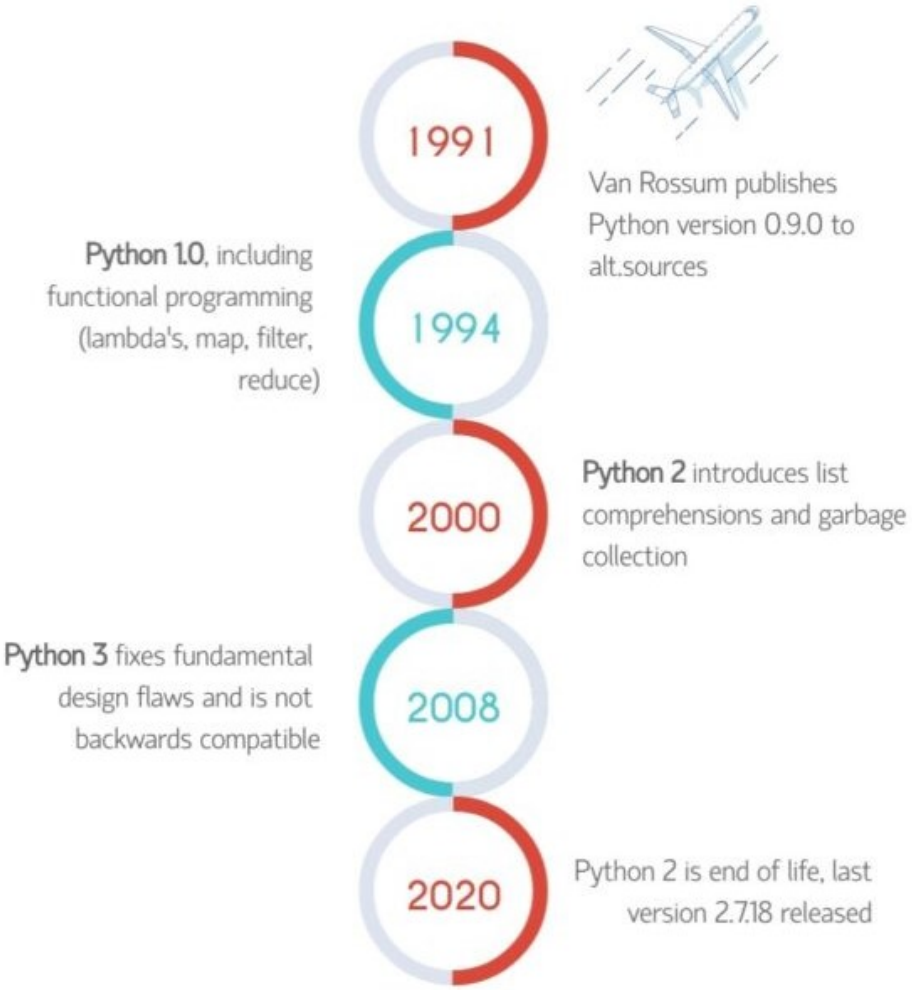
- Internet programming
- Database (DB) programming
- Text data processing
- Numerical operations
- Graphics
- Graphic User Interface (GUI)
- Distributed processing

And …

- Machine Learning
- Data Science
- Artificial Intelligence
- …

repl.it

# Timeline





https://youtu.be/ucD_1ryKKm0?si=OmayQnJJQ60lidZA


repl.it

Created by Daniel Benninger

# Elements of the Python Language

## What is  Character Set?

- **Character Set** is a bunch of  identifying elements in the programming language.

# Elements of the Python Language

## What is  Character Set?

- Letters: A-Z, a-z
- Digits: 0 to 9
- Special Symbols: space + - / ( ) [ ] = ! = < > , ‘ “ $ # ; : ? &
- White Spaces: Blank Space , Horizontal Tab, Vertical tab, Carriage Return.
- Other Characters: Python can process all 256 ASCII and Unicode Characters.

repl.it

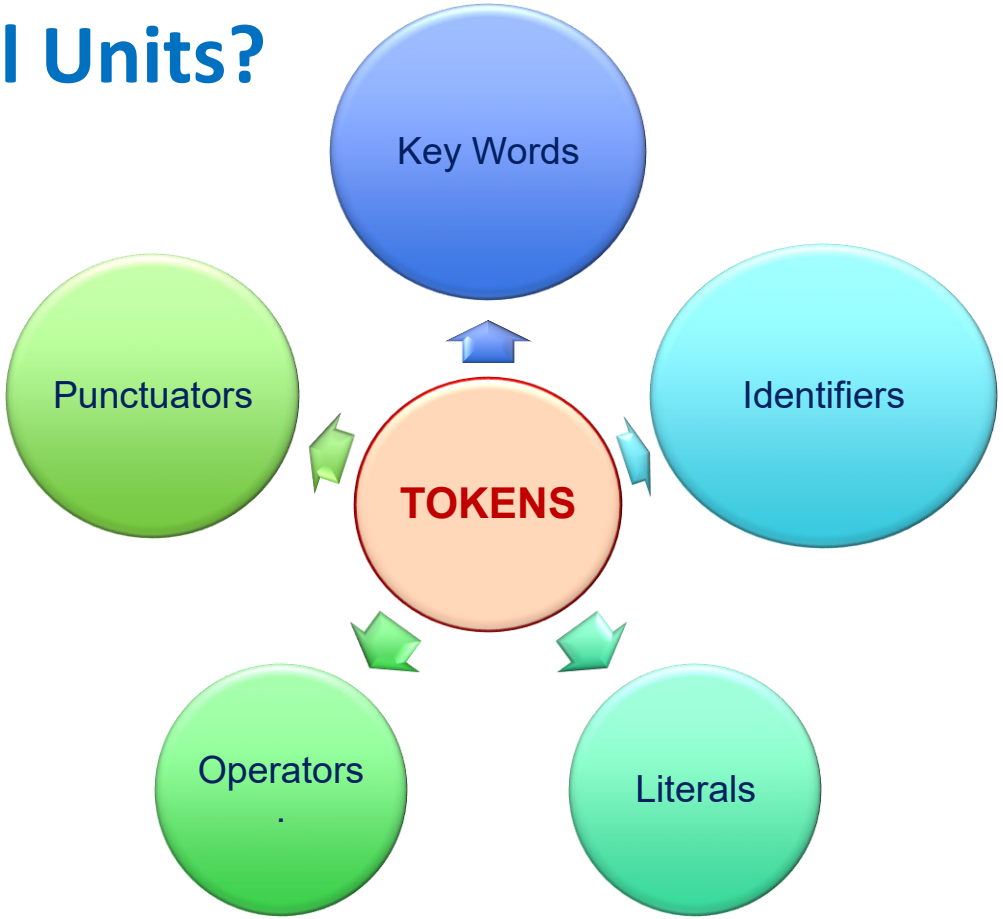
# Elements of the Python Language

## What are Tokens or Lexical Units?

- Individual  elements that are identified by programming language are called **tokens** or **lexical units**.

# Elements of the Python Language

## What are Tokens or Lexical Units?




repl.it

# Elements of the Python Language

## What are Key Words?

- **Keywords** are also called as reserved words these are having special meaning in python language.
- The words are defined in the python interpreter hence these can’t be used as programming identifiers.

# Elements of the Python Language

## Some Python Key Words

| | |
|---|---|
| and | or |
| break | class |
| continue | def |
| del | elif |
| else | except |
| exec | finally |
| for | from |

| | |
|---|---|
| global | if |
| import | is |
| not | with |
| pass | print |
| raise | return |
| try | while |

repl.it

# Elements of the Python Language

## What is an Identifier?

- A Python **Identifier** is a name given to a function, class, variable, module, or other objects that you'll be using in your Python program.
- In short, it's a name appeared in the program.

*For Example*: x,y,z, i,j,k, nr, firstname, last_name are valid identifiers

# Python Naming Conventions

1. An identifier can be a combination of uppercase letters, lowercase letters, underscores (_), and digits (0-9)
   *Hence, the following are valid identifiers: myClass, my_variable, var_1, and print_hello_world*
2. The first character must be letter
3. Special characters such as %, @ and $ are not allowed within identifiers
4. An identifier should not begin with a number
   *Hence, 2variable is not  valid, but variable2 is acceptable*

# Python Naming Conventions (cont.)

5. Python is a case-sensitive language and this behaviour extends to identifiers.
   *Thus, Labour and labour are two distinct identifiers in Python*
6. You cannot use Python keywords as identifiers
7. You can use underscores (_) to separate multiple words in your identifier
   *Hence, my_local_data_file is a valid identifier*

# Python Naming Conventions (cont.)

**Some valid identifiers**

Myfile1 DATE9_7_8 y3m9d3 _xs MYFILE _FXd

**Some invalid identifiers**

MY-REC 28dre break elif false del @home

# References

- Python Homepage
  - http://www.python.org
- Python Tutorial
  - http://docs.python.org/tutorial/
  - https://www.w3schools.com/python/
  - https://py-tutorial-de.readthedocs.io/de/python-3.3/ (auf Deutsch)
- Python Documentation
  - http://www.python.org/doc
- Python Library References
  - http://docs.python.org/release/2.5.2/lib/lib.html

# Introduction to Python Programming

## 1 - Output, variables and input.

repl.it

Resources created by Andy Colley (@MrAColley)

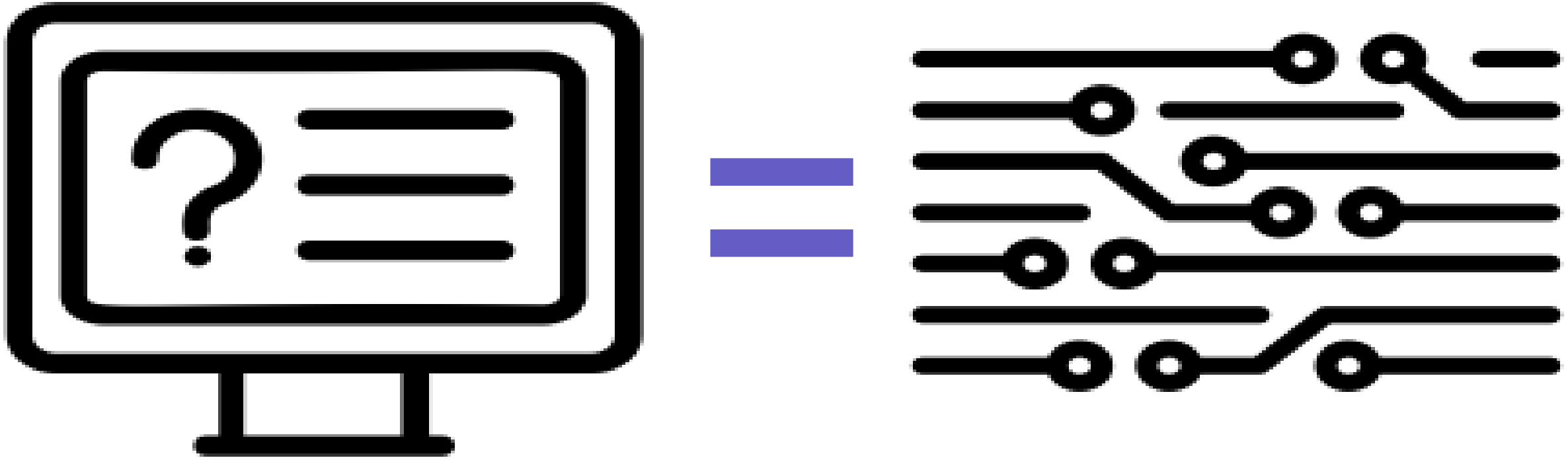
# Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

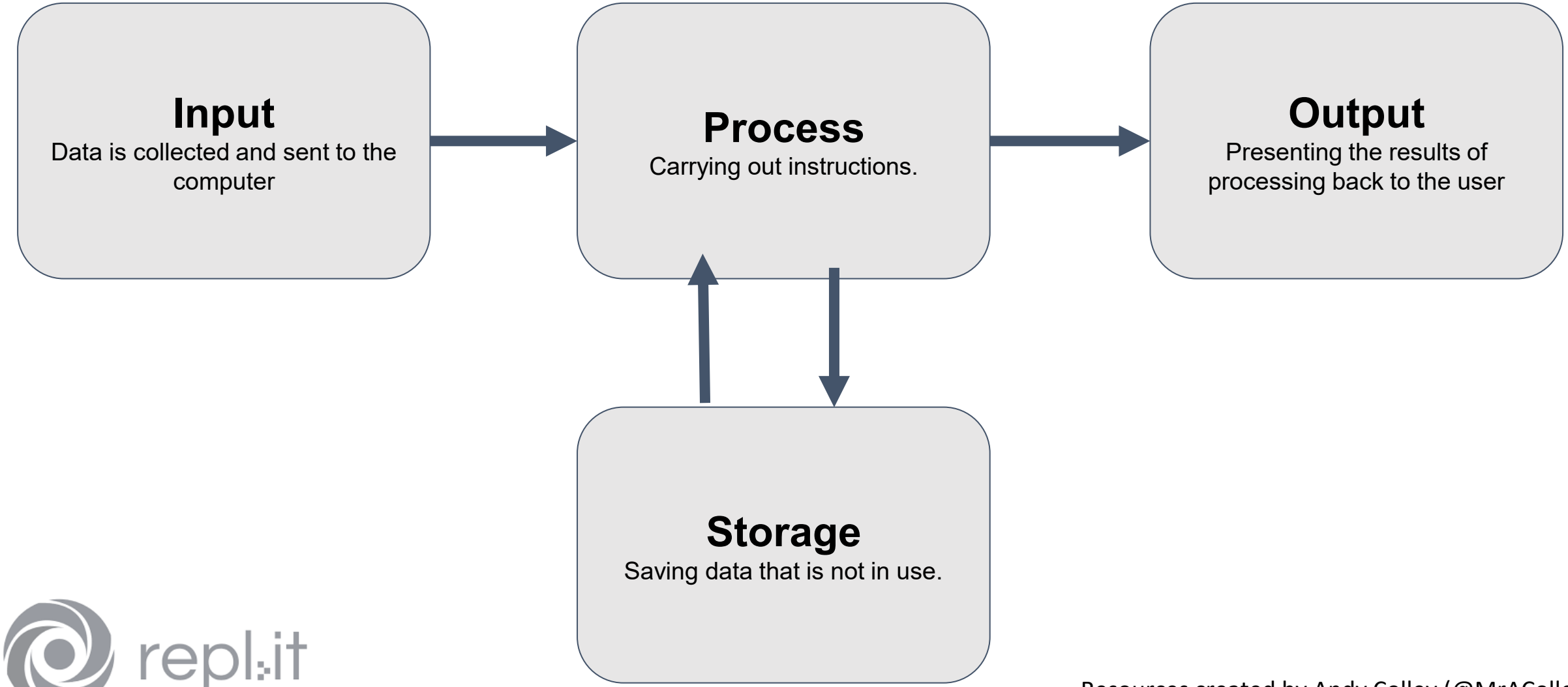
- Outputs text to the console
- Assigns data to variables
- Gets input
- Assigns input to variables.
- Refers to variables in other statements such as ‘print’

repl.it

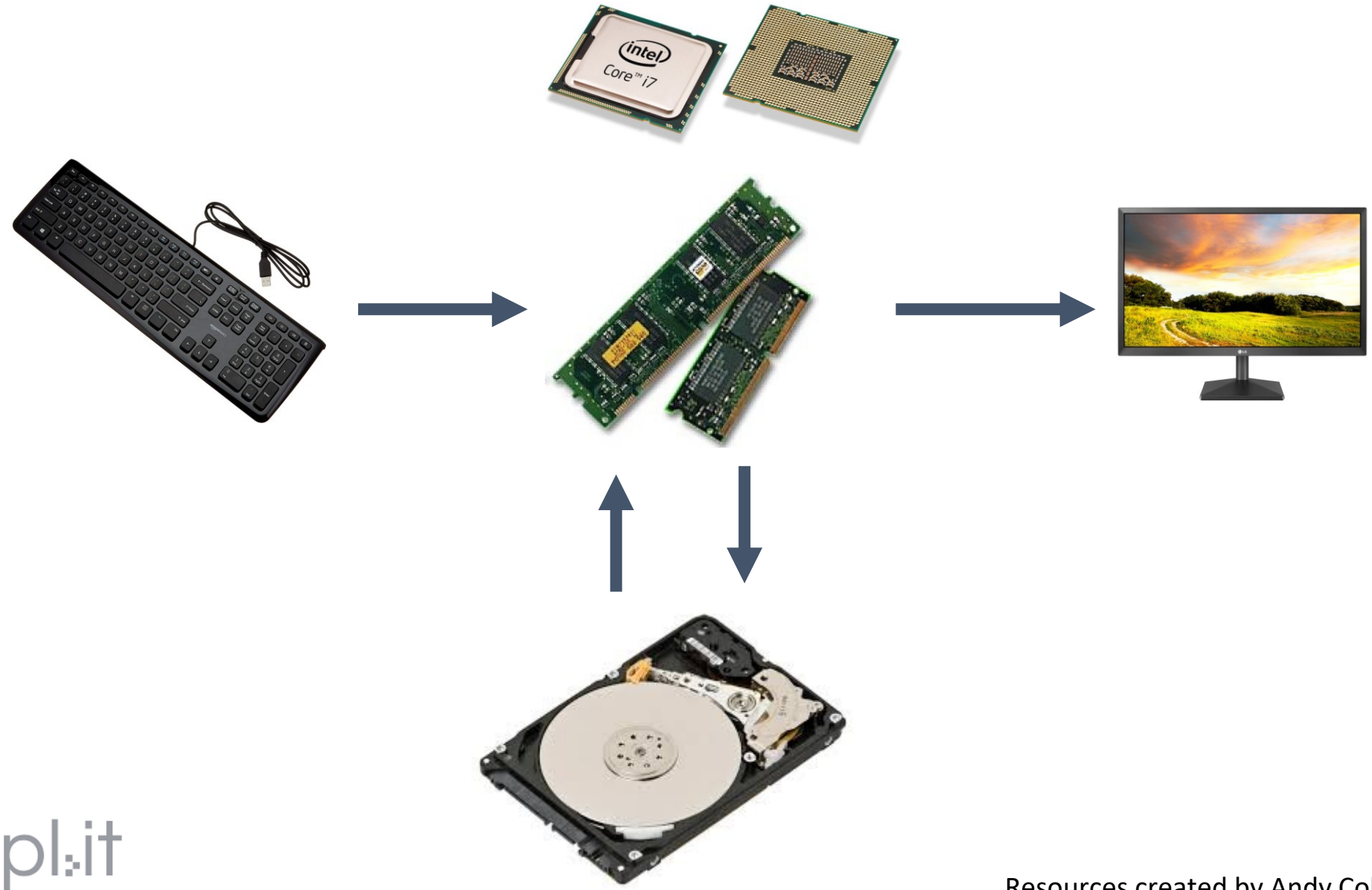
# What is a Computer?



repl.it


Input
Data is collected and sent to the computer
Process
Carrying out instructions.
Output
Presenting the results of processing back to the user
Storage
Saving data that is not in use.

repl.it




repl.it

Resources created by Andy Colley (@MrAColley)


## Input
Data is collected and sent to the computer

## Process
Carrying out instructions.

## Output
Presenting the results of processing back to the user

## Storage
Saving data that is not in use.

repl.it

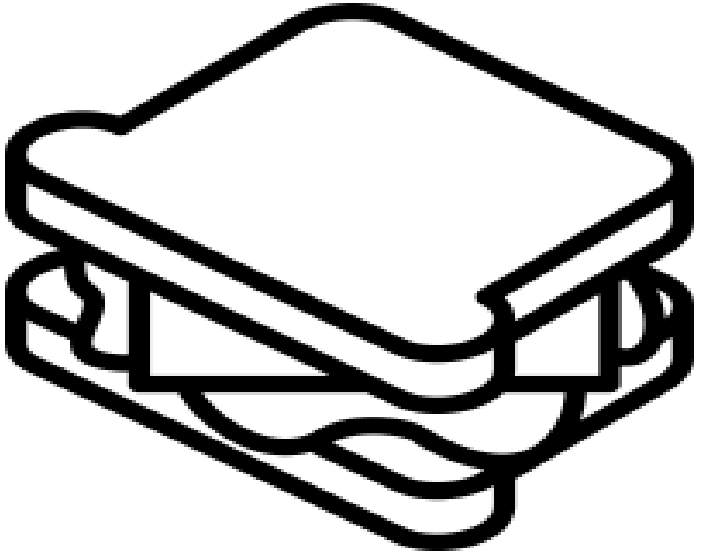
Resources created by Andy Colley (@MrAColley)

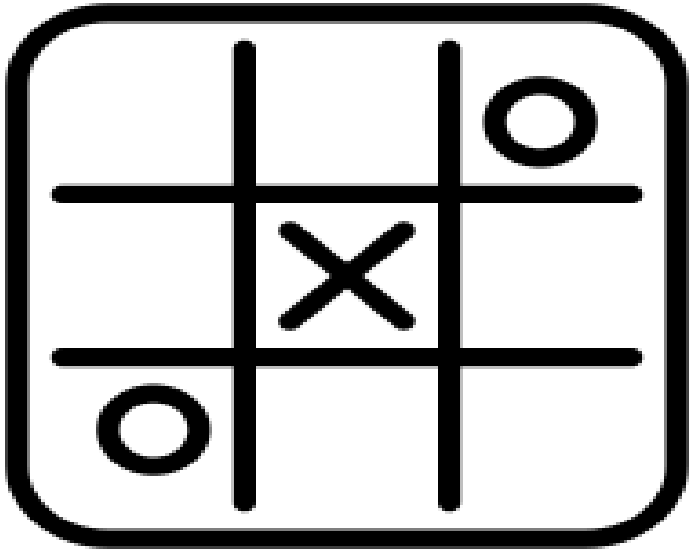
# Algorithms

*"An algorithm is a sequence of instructions for a task"*

Algorithm ≠ Computer Program

# Algorithms In The Real World





repl.it

# A Computer Program

Algorithms that are converted into code become computer programs.

## Algorithm

OUTPUT "Hello, World"

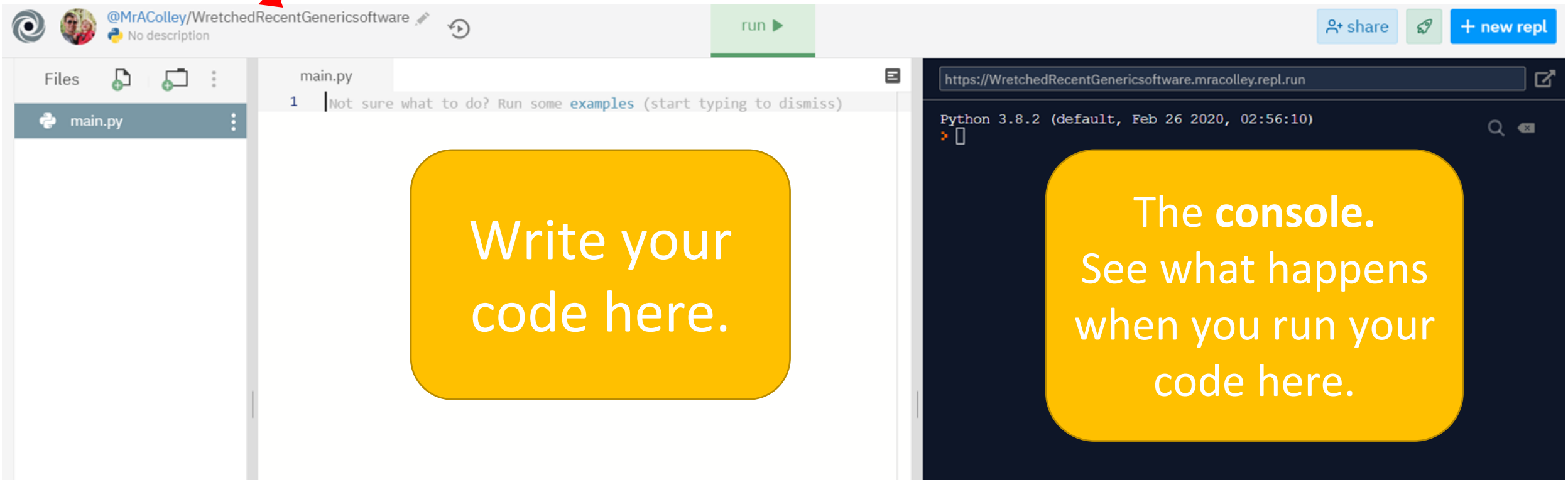
### Scratch


### Python

```
print("Hello, World!")
```

repl.it

Choose ‘Fork’ here




Write your code here.

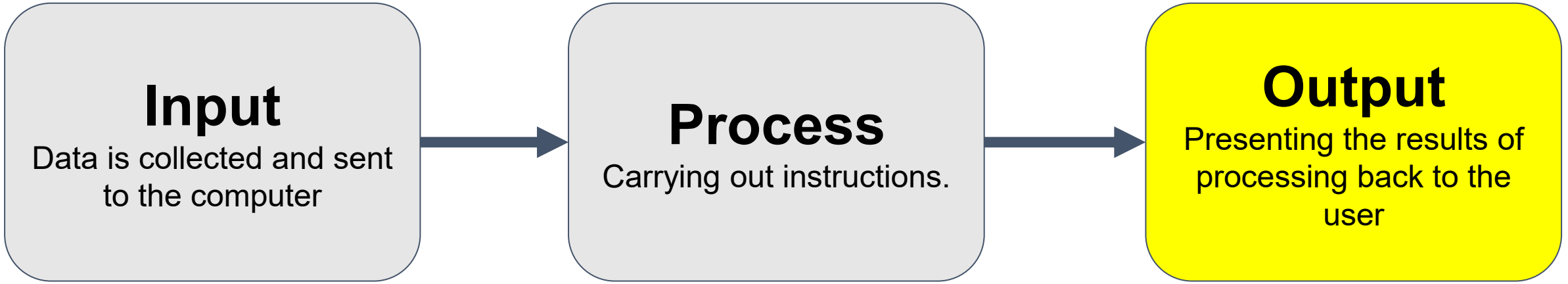
The **console.** See what happens when you run your code here.

repl.it

Resources created by Andy Colley (@MrAColley)

# OUTPUT

# Programming – Output




```
print("your text goes here")
```

repl.it

# Programming – Output - Investigate

```
print("your text goes here")
```

**Syntax – The Way The Code Is Written**

print – no capital p.
Brackets – after print and at the end.
Text **must** go in speech marks.

**Incorrect syntax = broken code that won't work = error!**

repl.it

# Programming – Output – Predict & Run

```
5   #Task 1 - Add a comment on line 7 to predict
    what the code on line 8 will do.
6
7
8   print("Hello World!")
```

Add comments to the code to predict exactly what it will output.

Run the code to see if you were correct.

repl.it

# Programming – Output – Investigate

```
print("Hello World!")

# Task Investigate

# What would the output of the code print("I love Computing") be?

# What would happen if the code print("I love Comping") was run?

# What would happen if the code print("I love Computing" was run?
```

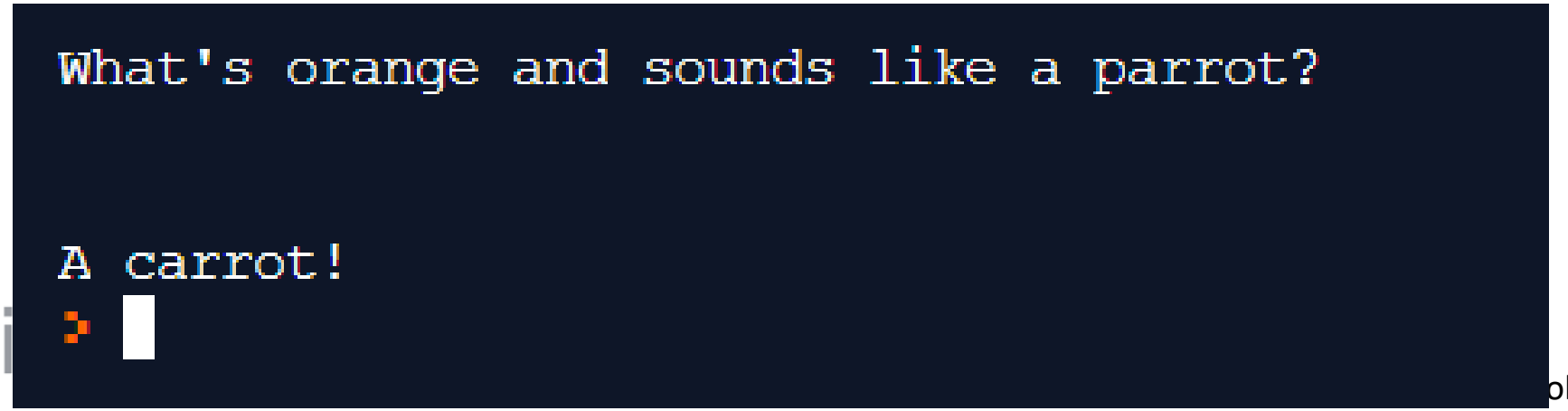
Add comments to the code to answer the questions.

https://repl.it/@MrAColley/11-Output

# Programming – Output – Modify & Make

**Modify** – reuse the print statement to add your own single line message to the program.

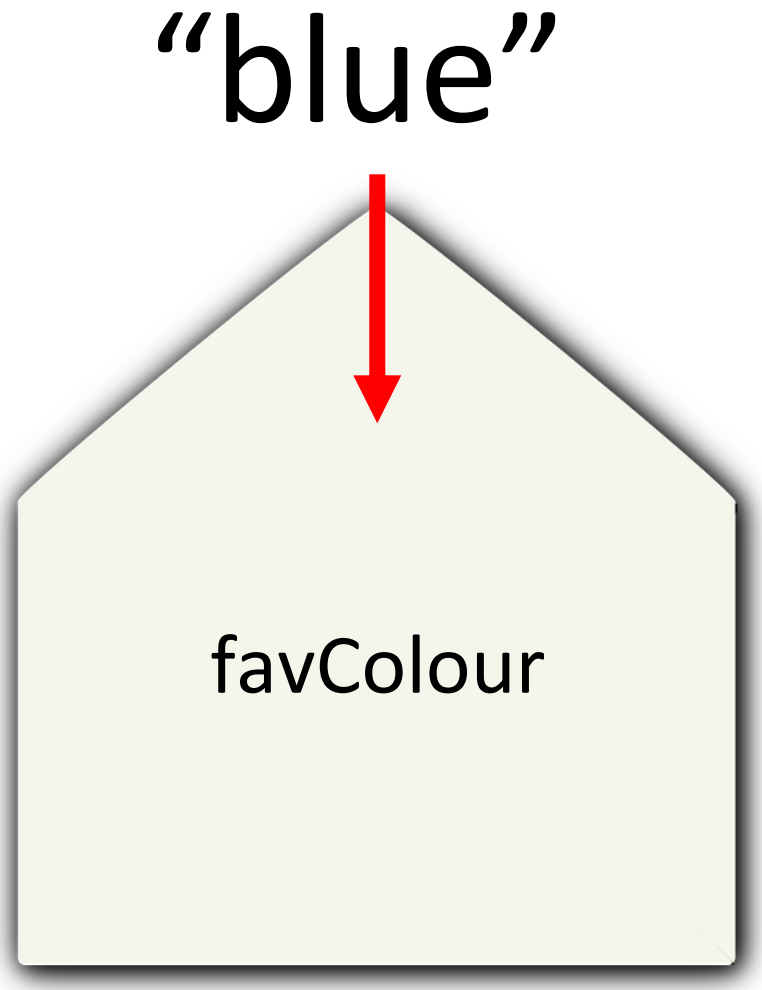
**Make** – use the print statement to output a joke that appears on multiple lines (keep it clean!). **Extra challenge** – can you figure out how to add blank lines like in my joke?



# VARIABLES

# Programming - Variables

- Placeholders for information.
- Can store one piece of information.
- Can be called whatever you like (keep it relevant)
- No spaces in the name.
- Name starts with a lower case letter.
- Can’t have 2 variables with the same name.
- Use the variable name whenever you want to
- output the information.




```
print(favColour)
print(“My favourite colour is “ + favColour)
```

repl.it

# Programming – Variables – Predict & Run

```
#Task 1 - Add comments to the code below to explain what it does

name = "Axl"

print(name)
```

Add comments to the code to predict exactly what it will output.

Run the code to see if you were correct.

repl.it

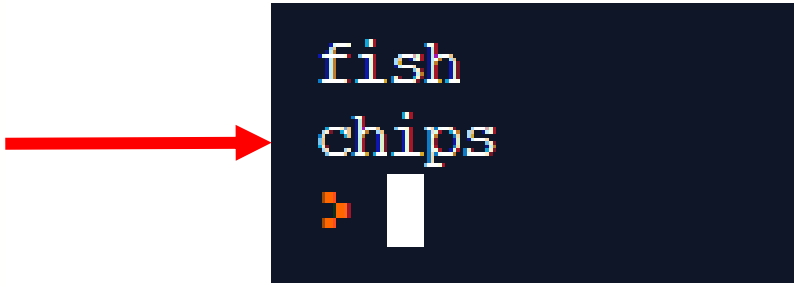
# Programming – Variables – Modify

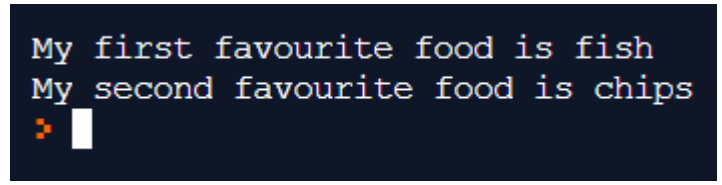
```
#Task 2 - Concatenation

#You can combine variables with strings in an output

name1 = "Axl"
name2 = "Slash"
#Add 2 more variables to store 'Duff' and 'Izzy'


#This line uses concatenation to join the variables together with the
string 'and' to make a sentence.

#Complete the line to output all of the variables
print(name1 + " and " + name2 + " and ")
```

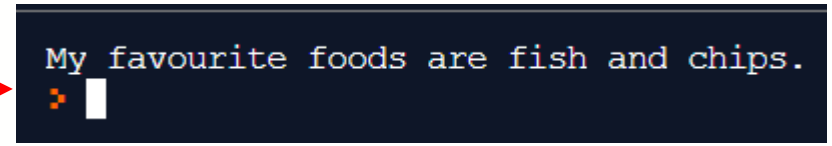
Resources created by Andy Colley (@MrAColley)

# Programming – Variables – Make

```
#Task 3

########################## SIMPLE ####################
# Assign your name and your favourite food to 2 separate variables.
# Output the contents of the variables on 2 separate lines

############## MEDIUM ##############################

# Output two sentences (not just the contents of the variables). The
first with your name, the second with your favourite food.


############## COMPLEX ##############################
# Output both pieces of information as part of the same sentence.
# Make sure that you have spaces and punctuation in the correct
places.
```



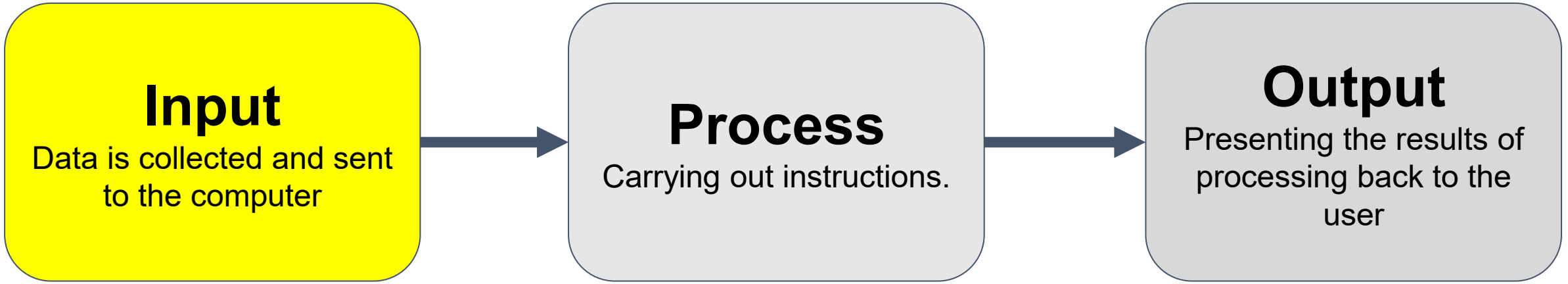




https://repl.it/@MrAColley/13-Variable-Assignment-Challenge

repl.it

# INPUT

# Programming – Input




```
print("What's your name?")
name = input()
print("Hello " + name)
```

repl.it

# Programming – Input – Predict & Run

```
#Task 1 - Add comments to the code below to
predict what will happen when it is run.

#Task 2 - add comments to the code to explain
what the variables are called and where they
are used. Make sure to show where the
variable.

print("Hi! What's your name?")

name = "Dave"

print("Hi " + name + "! How are you today?")
```

Add comments to the code to predict exactly what it will output.

Run the code to see if you were correct.

https://repl.it/@MrAColley/14-Input-With-Variables

repl.it

# Programming – Input - Efficiency

```
print("What's your name?")
name = input()
```

```
name = input("What's your name?")
```

# Programming – Input – Modify

```
# Task 3 - Adapt the code so that it assigns input into the 'name'
variable. CHALLENGE - put a prompt in the input command to ask the
user for their name.

# Task 4 - Combine lines 24 and 25  so that it the input has a prompt
in it.

#Task 5 - Adapt the output on line 26 so that it includes both the
name and the answer variables.

name = "Billy"
print("We want to know if you like progamming!")
print()
print("Do you like programming " + name + "?")
answer = input()
print("Great! You said " + answer + "!")
print("Let's learn some Python today")
```

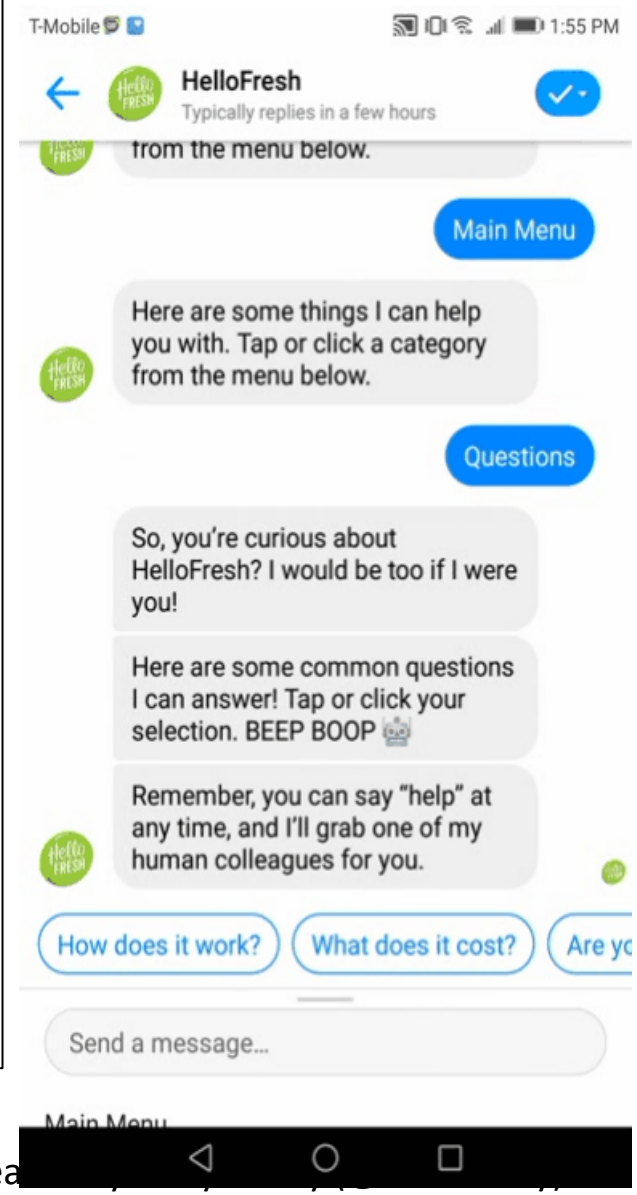
# Programming – Variables – Make Homework

**Chat-Bot Challenge**

Lots of websites use chat bots to interact with their customers. These chat bots are often very sophisticated and use AI to learn and adapt to the user. Our chat bot is going to be a bit simpler.

The chat bot should work like this:

- Ask the user their name and store it in a variable.
- Greet the user by name.
- Ask the user three questions about themselves and store their responses in three **different** suitably named variables.
- Respond to each of the questions one by one, using the user's name in the response.
- Output a summary of all of the user's answers in a single sentence.




repl.it

Resources crea

# Introduction to Python Programming

## 2 – Maths with Variables.


Resources created by Andy Colley (@MrAColley)

# Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create

Python code that:

- Performs simple maths (addition, subtraction, multiplication, division and integer division) using fixed numbers
- Performs simple maths using numbers stored in variables
- Converts strings to integers
- Gets number input and uses it in calculations

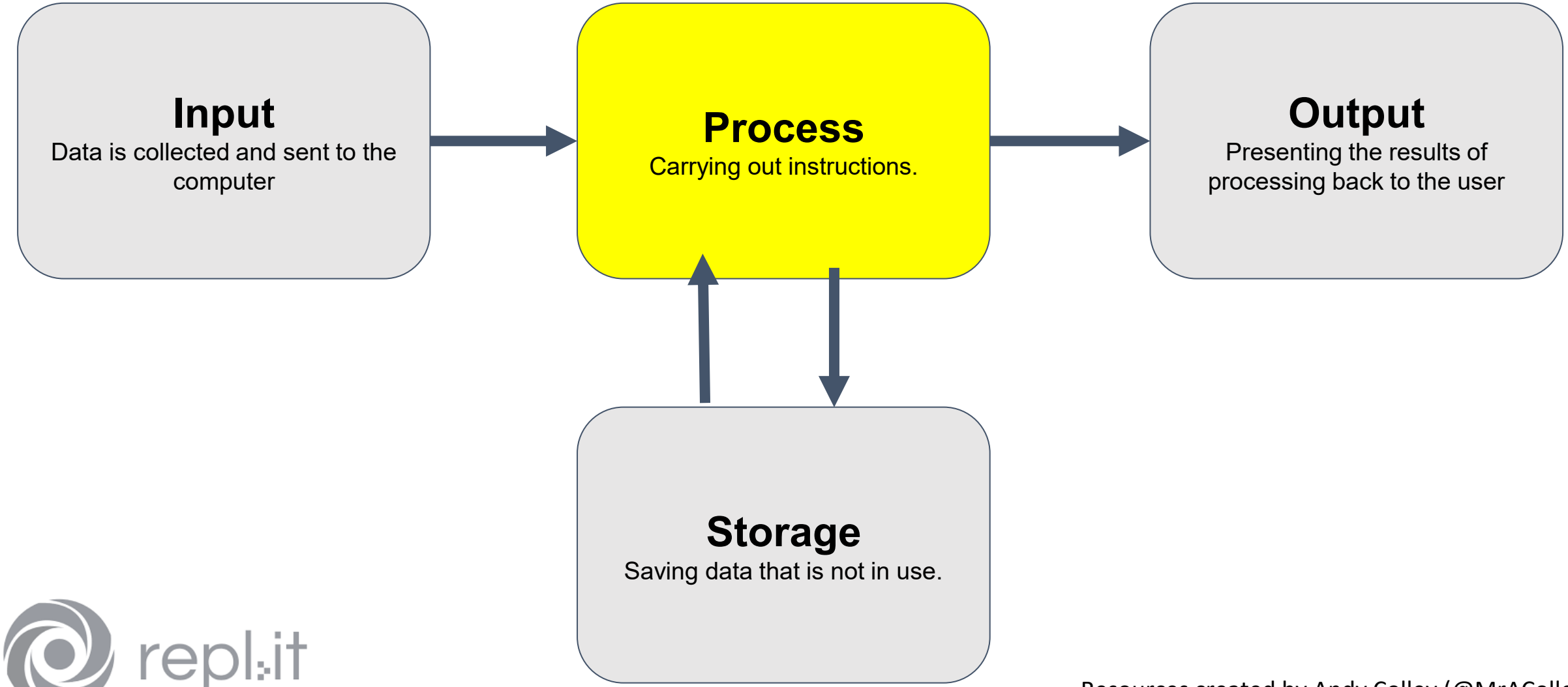
repl.it

# Maths - Operators

2 + 2

# Maths - Operators

```
+   addition
-   subtraction
*   multiplication
/   division (with decimal)
//  division (integer result)
```


Input
Data is collected and sent to the computer
Process
Carrying out instructions.
Output
Presenting the results of processing back to the user
Storage
Saving data that is not in use.

repl.it

# Programming – Operators - Predict & Run

```
# Task 1 - add comments to this code to predict what it will do.
print(8 + 2)

print( 8 - 2)

print(8 * 2)

print(8 / 2)

print(8 // 2)
```

https://repl.it/@MrAColley/21-Maths-part-1

repl.it

# Maths With Variables

```
num1 = 5
num2 = 10

result = num1 + num2

print(result)
```

Assign numbers to your variables.

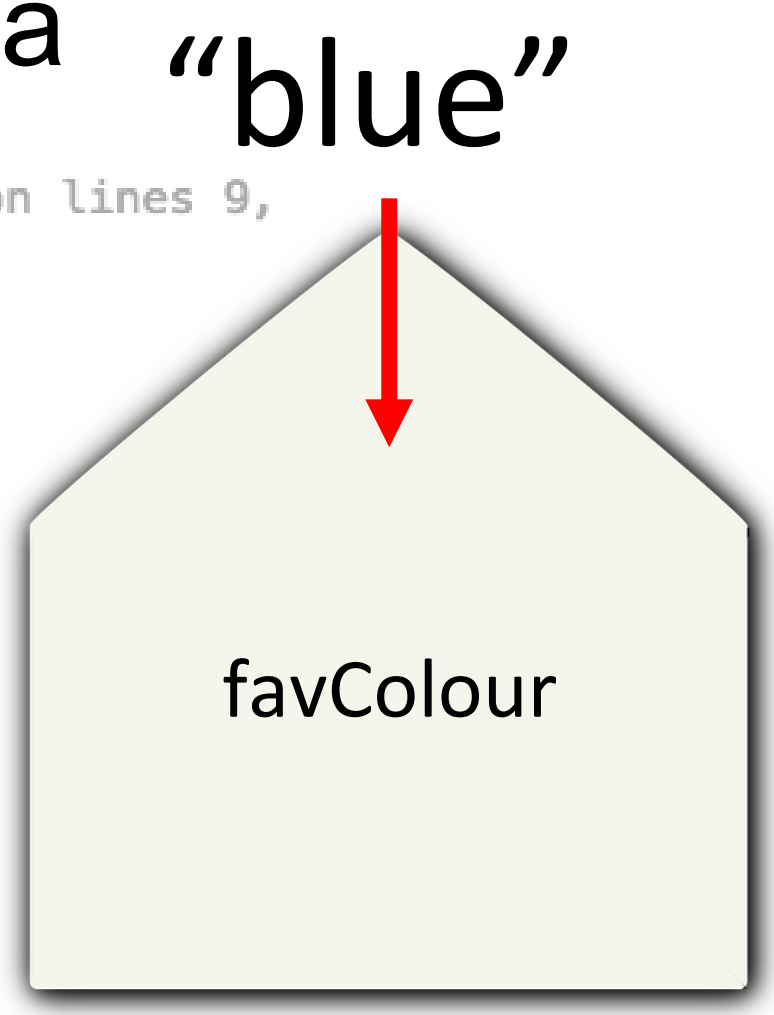
Perform the calculation. Assign the result to a **NEW VARIABLE.**

Don’t forget to output the result

repl.it

# Variables - vary - changing data

```
# Task1 – Add comments to the code to predict the output on lines 9,
10 and 14

name1 = "Ross"
name2 = "Monica"
name3 = "Joey"
name4 = "Rachel"
name5 = "Chandler"

print(name1 + " and " + name4)
print(name3)

name3 = "Phoebe"

print(name3)
```




repl.it

# Programming – Variable Maths – Modify & Make

```
score = 0
print(score)
score = 10
print(score)
```

repl.it

# Changing Variables With Maths

```
score = 0
print(score)
score = 10
score += 1
print(score)
```

```
score -= 1
print(score)
```

repl.it

# Programming – Input & Variables - Integers

`Input` works with strings. Strings are **text.**

Computers can't do maths/logic with text. **They have to use numbers.**

Whole numbers are called **integers** (ints for short).

We have to write code to convert our string input to an int before we can do maths with it.

# Programming – Input & Variables - Integers

```
num1 = int(input("Enter a number"))
```

Tells the computer to **convert to an integer**

**BRACKETS!** Because syntax!

repl.it

# Programming – Input & Variables - Integers

```
print(str(num1)
```

Tells the computer to
**convert an integer variable to a string**

repl.it

# Task 5 - Homework Challenge - Area Calc

Create a program that allows the user to enter 2 numbers representing the width and length of a rectangle.

The program calculates and displays the area of the rectangle.

Example solution - https://repl.it/@MrAColley/25ChallengeExampleSolution

# Extra Credit Challenges

### Perimeter Calc

Create a program that allows the user to enter 2 numbers representing the width and length of a rectangle. The program calculates and displays the perimeter of the rectangle.

### Restaurant Tip Calculator

Create a program that allows the user to enter the price of a meal at a restaurant. The program calculates the amount of the tip to be paid at 20%. The tip and total price are then displayed separately.

### Volume and Surface Calc

Create a program that allows the user to enter 3 numbers representing the height, width and length of a cuboid. The program calculates and displays the volume and total surface area of the cuboid.

# Introduction to Python Programming

## 3 – Selection using IF, ELIF and ELSE.

# Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

- uses Boolean conditions
- uses selection using **IF**, **ELIF** and **ELSE** for more than two situations

# Selection - Three Or More Outcomes

```
if weather == “rain”:
        print(“Take your umbrella”)
else:
        print(“No special advice for you”)
```

If the condition is **false**
then skip to the **else** and do that instead.

What if we want to handle more weather conditions?

# elif

Add elif with a condition **between if and else**.

You can add as many elifs as you need

# Selection with 3 or more outcomes - **The algorithm**

1. Start with an **if**
   → set the first condition

2. Add as many **elifs** as you need
   → give each one a new condition

3. Finish with an **else**
   → no condition needed

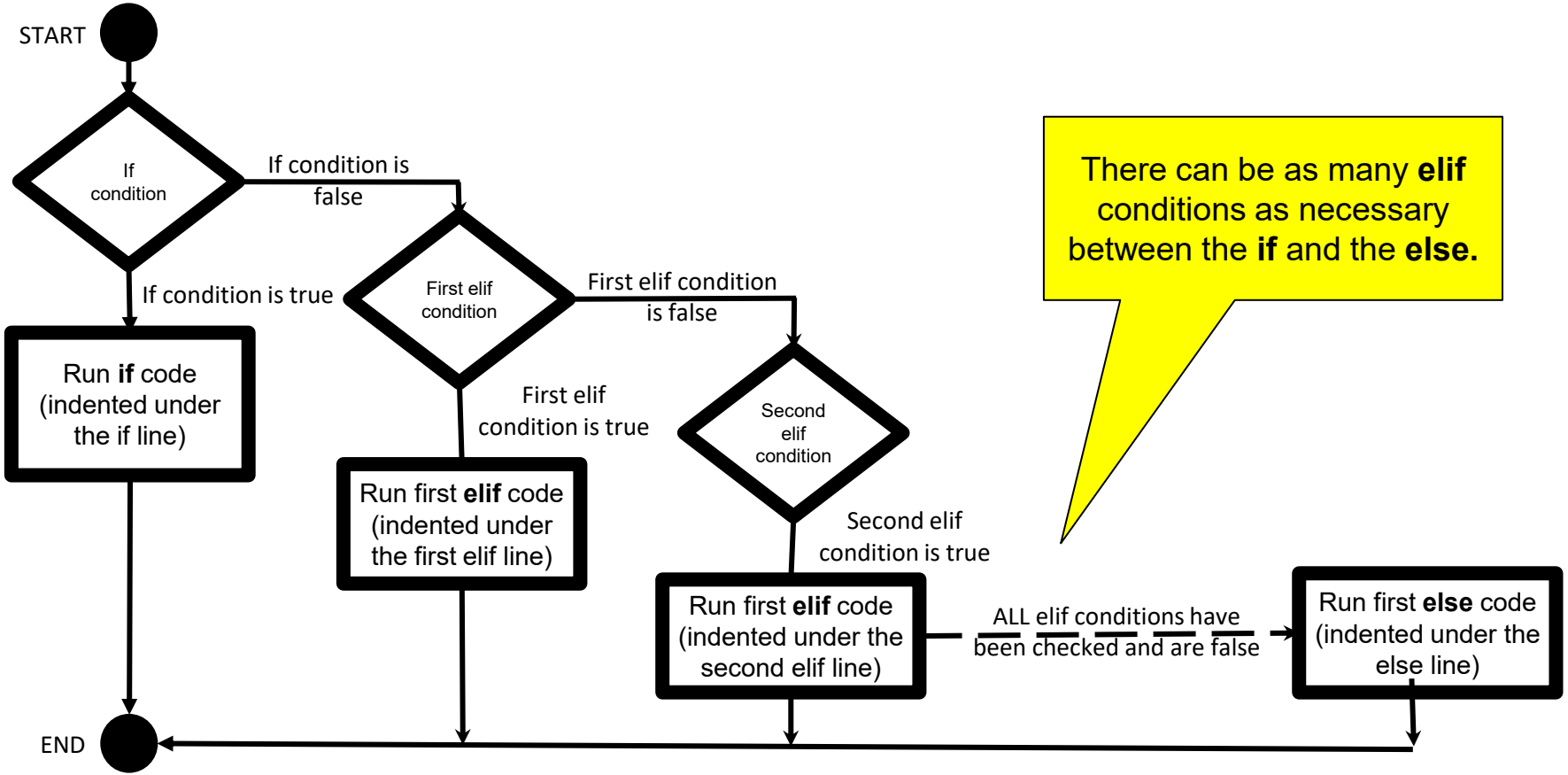
# Selection - Three Or More Outcomes

```
if weather == “rain”:
        print(“Take your umbrella”)
elif weather == “snow”:
        print(“Take your scarf”)
else:
        print(“No special advice for you”)
```

# Selection - Three Or More Outcomes

```
if weather == “rain”:
        print(“Take your umbrella”)
elif weather == “snow”:
        print(“Take your scarf”)
elif weather == “sunny”:
        print(“Take your sunglasses”)
else:
        print(“No special advice for you”)
```

# Selection With Three Or More Outcomes - **Flowchart**



# Selection With Three or More Outcomes - Coding Tips

1. Start with if.

2. Write a condition for your first outcome.

3. DON'T FORGET THE COLON after the condition.

4. Code that should run if the condition is **true.**
DON'T FORGET TO INDENT (use the **tab** key)

```
if num1 > num2:
    print (str(num1) + " is bigger.")
elif num1 < num2:
    print (str(num2) + " is bigger")
else:
    print("The numbers are the same")
```

5. Add an elif.
Write a condition to check for your second outcome.

6. Keep adding elifs with conditions until there is only one condition left.

7. Add an else for your final outcome.
No condition needed.

# Selection With Three Or More Outcomes

**Task - Which Room?**

- Write a program that asks the user for their name and which subject they are studying.
- The program should output a message telling the student by name which room to go to for that class (make up the room numbers if you need to). You should include at least 3 subjects and have a message such as 'I don't know which room that class is in' for any you don't include.

***Example: An input of 'Ben' and 'Computing' might get an output of 'Hi Ben, go to room 401 for Computing'***

# Extra Challenge - The INSULT-O-MATIC 5000!!!!

Write a program that:

- Asks for the user's name.
- Asks the user to input a number between 1 and 5.
- Outputs a personalised insult (that includes the user's name) depending on which number they picked.

Keep your insults clean!

# Introduction to Python Programming

## 4 – Iteration using WHILE loops.


Resources created by Andy Colley (@MrAColley)

# Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

- Iterates using **WHILE** loops
- Sets Boolean **conditions** to trigger loops
- Uses **counters** in loop conditions

repl.it

# Boolean Operators - Used in Conditions

| | |
|---|---|
| == | Equal to/The same as |
| != | Not equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |

# Boolean Operators are used to compare TWO pieces of data

data1 ***boolean operator*** data2

# Iteration

Starts the iteration statement.

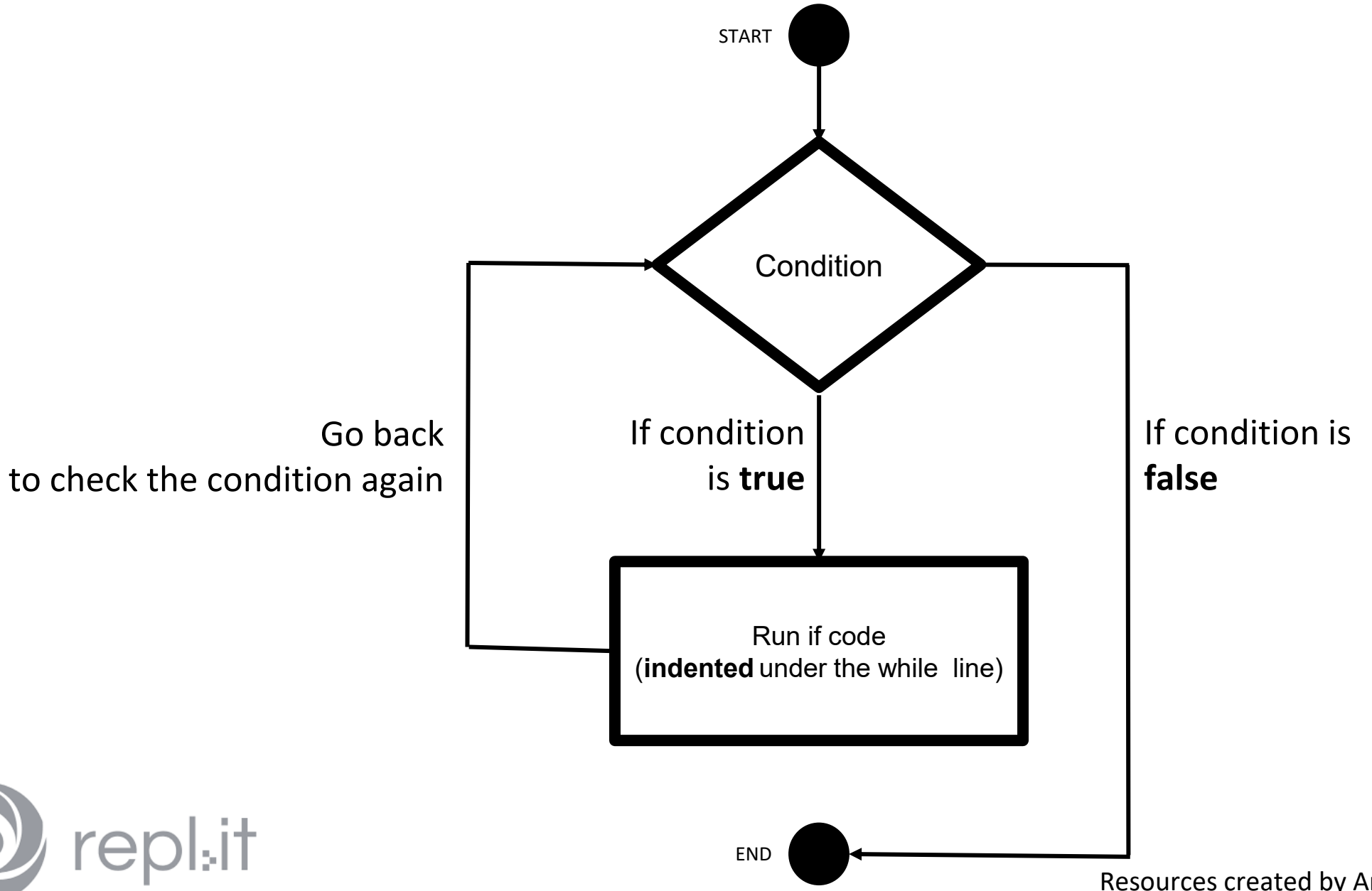
Colon - because syntax

```
while boolean condition:
    do this
```

This instruction is repeated while the condition is TRUE.

Indented - use the tab key.
Your code won't work if you don't indent inside the iteration statement.

# Iteration With While - **Flowchart**




repl.it

Resources created by Andy Colley (@MrAColley)

# Iteration with while - **Coding Tips**

Not part of the loop (comes before 'while').
Runs **before the loop starts.**

```
answer = input("What is the capital of France?")
while answer != "Paris":
    print("Incorrect! Try again.")
    answer = input("What is the capital of France?")

print("Correct!")
```

1. Start with while

2. Write the Boolean condition
(usually what you DON'T want the user to input).

3. DON'T FORGET THE COLON after the condition.

4. Code that is repeated while the condition is **true.**
DON'T FORGET TO INDENT (use the **tab** key)

Not part of the loop (not indented).
Runs **after the loop ends**.

# Iteration - Tasks Part 2

**Task**

Write a program that stores a secret number in a variable
(you decide the number and the name of the variable)

The user has to guess the secret number, the program should loop until they get it right.

Once the user has guessed correctly they get a congratulations message

repl.it

# Iteration With A Counter

```
counter = 1

while counter < 5
    print(“Hello”)

print (“The loop has ended”)
```

repl.it

# Iteration With A Counter

```
counter = 1

while counter < 5
    print(“Hello”)
    counter +=1

print (“The loop has ended”)
```

# Iteration With A Counter

```
counter = 1

while counter < 5
    print(“The counter is ” + counter)
    counter +=1

print (“The loop has ended”)
```

Add to the counter as
**the last command inside the loop**

repl.it

# Iteration With A Counter - Tasks Part 1

```
# Task 1

  # Add comments to explain what the counter variable is used for, and what +=1 does.
  # Add a comment that explains what the condition checks for.
  # Add a line of code inside the loop that outputs the counter variable every time the loop runs.  What do
  you notice about what happens to it each time the loop repeats?
  # Add a line of code after the loop that outputs the message 'The loop has finished'

counter = 1

while counter < 5:
  print("This code is inside the loop")
  counter += 1


# Task 2

  # Add a line of code to the loop that subtracts one from the counter every time the loop runs
  # Add a line of code after the loop that outputs 'Blast off'
  # EXTRA CHALLENGE - Adapt the code so that the user inputs the start value of the counter variable.

counter = 10

while counter > 0:
  print(counter)
```

repl.it

Resources created by Andy Colley (@MrAColley)

# Iteration With A Counter - Tasks Part 2

## Task

# Write a program that uses a loop to output the seven times table up to 12 x 7
*(HINT - use the counter variable and combine it with what you learned about maths with variables.)*

# Output the multiplied number as part of a sentence. *Example - '1 times 7 is 7'*

# EXTRA CHALLENGE - ask the user to input a number and output the multiplication table for the input.

repl.it

# Homework Challenge - Cubes Cubes Cubes

The cubed number sequence starts: 1, 8, 27, 64, 125.

Write a program that:

- Asks the user to input a number
- Display N numbers in the cubed sequence according to user input.

repl.it

# Introduction to Python Programming

## 5 – Using LISTS.

repl.it

Resources created by Andy Colley (@MrAColley)

# Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

- Outputs one item from a **list**
- Outputs a whole list
- Changes an item in a list
- Adds an item to a list
- Removes an item from a list

repl.it

# Variable or List?

**Variable** - stores **one** piece of data with an identifier.

```
player1 = Mary
player2 = Sean
player3 = Atif
```

**List** - stores **more than one** piece of data with the same identifier.

```
players = [“Mary”, “Sean”, “Atif”]
```

repl.it

# Creating a List - How to Code

1 - **Name** the list. Use camelCase if it's more than one word.

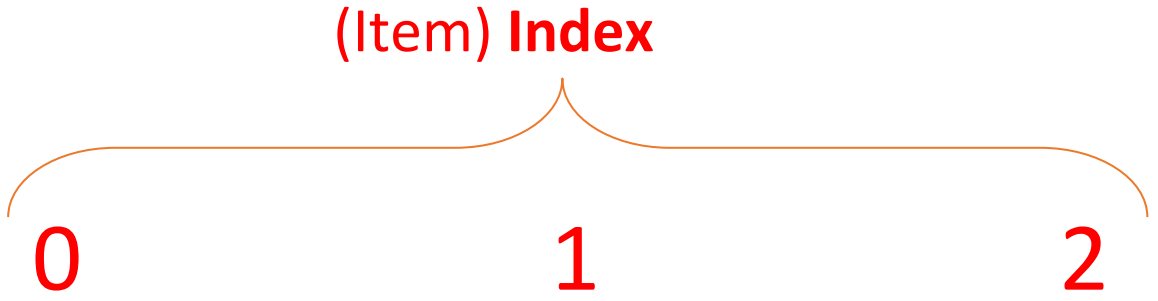
```
players = ["Mary", "Sean", "Atif"]
```

2 - The = symbol **assigns** data into the array

3 - All **the items in a list** are surrounded by square brackets with a comma between each one.

repl.it

# Identifying One Item In A List




```
players = ["Mary", "Sean", "Atif"]
```

repl.it

# Output From A List

# Output One Item From A List

```
players = [“Mary”, “Sean”, “Atif”]


print(players[2])
```

1 - Use the **print** statement.

2 - Put the **name of the list** inside the normal brackets.

3 - Put the **index** (number) of the item you want to output **in square brackets.**

repl.it

# Lists and Output

```
players = [“Mary”, “Sean”, “Atif”, “Steve”, “Lucy”]
```

```
print(players[5])
print(players[3])
print(players[1])
print(players[3] + players[4])
```

repl.it

# Task - List Output

```
fruit = ["Apple", "Banana", "Grape", "Strawberry", "Melon", "Orange"]

# Task 1
  # Add comments to predict what the following lines of code will do.

  # Alter the fourth print command so that it outputs a vaild item from the list that
  hasn't yet been used.

print(fruit[3])

print(fruit[5])

print(fruit[0] + " " + fruit[2])

print(fruit[6])

# Task 2
  # Write code to output the whole list - you should be able to do this with one line of
  code.abs


# Task 3
  # Ask the user to input a number between 0 and 5.  Output the item in the list that
  matches the number they have input.
```

repl.it

# Change & Edit Items in a List

# Change One Item In A List

```
players = [“Mary”, “Sean”, “Atif”]

players[0] = “Bill”
```

1 - The **item (index)** in the list to be replaced

2 - The = symbol used for **assignment**.

3 - The new data to go into the list (at the index position.

repl.it

# Lists and Assignment

```
players = [“Mary”, “Sean”, “Atif”, “Steve”, “Lucy”]
```

```
players[1] = “Oliver”

players[4] = “Jane”

players[2] = players[3]
```

repl.it

# Task - List Assignment 1

```
# Task 1
countries = ["UK", "USA", "Chad", "Australia", "Thailand"]

# Add comments to the code to explain what the following lines do.

countries[3] = "Mexico"

countries[0] = "Iceland"

countries[1] = countries[4]

# Add comments to predict what the list looks like now.

# Add a line of code to print the whole list and check your prediction
```

repl.it

# Task - List Assignment 2

```
# Task 2

squareNumbers = [1,4,9,16,25,36]

# Add comments to explain what the following lines of code do.

squareNumbers[5] = 49

squareNumbers[0] +=1

total = squareNumbers[3] - squareNumbers[1]

# Add comments to predict what the list looks like now.

# Add a line of code to print the whole list and check your prediction
```

repl.it

# Add & Remove Items From A List

# Add & Remove From A List

```
players = [“Mary”, “Sean”, “Atif”, “Steve”, “Lucy”]
```

```
players.remove(“Sean”)
players.pop()

players.append(“Dave”)
players.insert(2, “Julia”)
```

repl.it

# Task - Add & Remove From A List 1

```
# Task 1

food = ["bacon", "cheese", "pasta", "beans"]

# Add comments to explain wht the following lines of code do.

food.append("tomatoes")

food.insert(1, "ice cream")

food.remove("cheese")

food.pop()

# Add a comment to predict what the list looks like open

# Write code to print the whole list.  Was your prediction correct?
```

repl.it

# Task - Add & Remove From A List 2

```
videoGames = ["Mario", "Sonic", "Joust", "Zelda"]

#Task 2

# Write code to perform the following tasks.

  # Add 'Minecraft' to the start of the list.

  # Ask the user to input a number between 0 and 4 and store it in a variable.
  Output the item at this position in the list.

  # Ask the user to input the name of a video game and store it in a variable.
  If this video game is in the list then remove it from the list.  If it isn't
  in the list then add it to the end.
```

repl.it

# Task - Independent Challenge

- Create an array called 'names' that **stores five names** in it (you choose the names).
- Ask the user what their name is. Store their input in a variable.
- Ask the user to enter a number between 0 and 4. Store their input in a variable.
- Replace the data at the position that matches the number entered by the user in the names array with their name.

repl.it

# Find An Item In A List

# Find An Item In A List

```
players = [“Mary”, “Sean”, “Atif”, “Steve”, “Lucy”]
```

```
if “Atif” in players:
    run this code
else:
    run that code
```

# Homework Challenge - Beat The Zombie!

- Create a list of possible weapons.
- In a variable called ‘zombieWeakness’ store the name of one of the weapons from the list.
- Output messages telling the user that they have encountered a zombie and should prepare to fight.
- Output the list of weapons to the user.  Ask if they want to type 1 to use one from the list or 2 to pick their own.  If they type 1 then they should input the weapon name - store it to a new variable. If they type 2 they should input the weapon name - add it to the list and save it to a new variable.
- If the weapon picked matches the zombieWeakness, output a message telling the user that they have won the fight.  Otherwise output a message saying that they have lost.

repl.it

# Introduction to Python Programming

## 6 - SUBROUTINES.


Resources created by Andy Colley (@MrAColley)

# Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

- Defines a **subroutine**
- Calls a subroutine
- Creates a subroutine that uses arguments
- Gets user input and use it as an argument in a subroutine
- Returns a value from a subroutine

repl.it

# What Is A Subroutine?

- A subroutine gives a single name to a set of actions.
- You create a subroutine by **defining it.**
- You can use the subroutine at any time in your program by **calling** it.

1 - **Define** the subroutine and give it a name.

```
def say_hi():
    print("Hello there!")


say_hi()
```

2 - **Add the code** that you need to complete the task.
*Indent code statements!*

3 - **Call** the subroutine whenever you need it by typing its name.

repl.it

# Naming Subroutines

- Subroutine names **do not** use camelCase
- They use all **lower case** with **underscores** between the words.
- This helps us tell the difference between subroutines and variables/lists when we are reading the code.

```
say_hi()
add_one()
get_input()
```

repl.it

# Tracing Subroutines

Defining subroutines

Subroutines are **defined** at the top of your program.

But they do not run until they are **called** in the main program

```
1   def say_hi():
2       print("Why hello there!")
3
4   def offer_drink():
5       print("Would you care for a spot of tea?")
6
7   def offer_food():
8       print("Biscuit?")
9
10  def say_bye():
11      print("Cheerio then.")
12
13
14  offer_drink()
15  say_hi()
16  offer_food()
```

Main program.
Use subroutines

# Tracing Subroutines

Write down the line numbers in order that they will execute when the program is run.

```
1   def say_hi():
2     print("Why hello there!")
3
4   def offer_drink():
5     print("Would you care for a spot of tea?")
6
7   def offer_food():
8     print("Biscuit?")
9
10  def say_bye():
11    print("Cheerio then.")
12
13  print("Welcome to the hospitality program!")
14  say_hi()
15  print("what's your name?")
16  offer_drink()
17  print("Oh, lovely")
18  offer_food()
```

repl.it

# Subroutines That Return A Value

# Subroutines That Return a Value (Functions)

- A subroutine can **return** (send) **some data back to the main** (calling) **program**.
- When you do this, you should store the returned value in a variable in the main program.

1 - **Define** the subroutine and give it a name.

```
def adder():
        num1 = 10
        num2 = 15

        return num1 + num2

outputNum = adder()

print(outputNum)
```

2 - Add the code that you need to complete the task.

3 - Use **return** followed by the task that you want to perform

3 - **Call** the subroutine in the main program (NOT INDENTED).

*Look at how the subroutine is assigned to a variable.*

repl.it

# Subroutines That Use Arguments

# Subroutines With Arguments

- We can put data into a subroutine.  To do this we use **arguments** (aka *parameters*)**.**
- You can think of arguments like variables used by the subroutine.

```
def add_five(num1):
    print(num1 + 5)

add_five(42)
```

1 - The argument name goes in brackets after the subroutine name.

2 - Put the actual data in the brackets when you call the subroutine.
This will be put into the *num1* argument and used by the subroutine.

repl.it

# Subroutines With Arguments

- We can get **input from users** and use that as arguments too.

```
def add_five(num1):
 print(num1 + 5)


userInput = int(input("Enter a number"))
add_five(userInput)
```

1 - Get the user to input and save it in a variable

2 - Use the variable as the argument. This will put the data from the userInput variable into the num1 argument.

# Subroutines With Arguments

- We can get **input from users** and use that as arguments too.

```
def add_five(num1):
 print(num1 + 5)


add_five(int(input("Enter a number")))
```

# Independent Challenge - Calculator

- Define four subroutines - **add, subtract, multiply, divide** that add multiply etc two numbers and return the result.
- Each should have two integer number arguments.
- The user is asked to input two numbers.
- These numbers will be passed as arguments into one of the subroutines.
- The user is asked to input 1 for add, 2 for subtract etc.
- If they input 1, call the ‘add’ subroutine, input 2 calls the ‘subtract’ subroutine etc
- Output the returned result as part of a sentence.

repl.it

# Introduction to Python Programming

## Packages + Plotting

# Modules

- As program gets longer, need to organize them for easier access and easier maintenance
- Reuse same functions across programs without copying its definition into each program
- Python allows putting definitions in a file
  - use them in a script or in an interactive instance of the interpreter
- Such a file is called a **module**
  - definitions from a module can be *imported* into other modules or into the *main* module

- A module is a file containing Python definitions and statements
- The file name is the module name with the suffix .py appended

# Modules Example
## Definition

fib.py - C:\Users\karkare\Google Drive\IITK\Courses\2016Python\Programs\fib.py (2.7.12)

File Edit Format Run Options Window Help

```
# Module for fibonacci numbers

def fib_rec(n):
    '''recursive fibonacci'''
    if (n <= 1):
        return n
    else:
        return fib_rec(n-1) + fib_rec(n-2)

def fib_iter(n):
    '''iterative fibonacci'''
    cur, nxt = 0, 1
    for k in range(n):
        cur, nxt = nxt, cur+nxt
    return cur

def fib_upto(n):
    '''given n, return list of fibonacci
    numbers <= n'''
    cur, nxt = 0, 1
    lst = []
    while (cur < n):
        lst.append(cur)
        cur, nxt = nxt, cur+nxt
    return lst
```

fib.py

# Modules Example
## Usage

```
>>> import fib
>>> fib.fib_upto(5)
[0, 1, 1, 2, 3]
>>> fib.fib_rec(10)
55
>>> fib.fib_iter(20)
6765

>>> fib.__name__
'fib'
```

Within a module, the module's name is available as the value of the global variable __name__.

# Modules Example
## **Usage** (cont.)

To import **all** functions from a module, in the current symbol table

```
>>> from fib import *
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(8)
21
```

# Packages (Libraries)

- A Python **package** (*library*) is a collection of Python modules

- Packages are a way of structuring Python's module namespace by using dotted module names
  - The module name A.B designates a submodule named B in a package named A.
  - The use of dotted module names saves the authors of multi-module packages like *NumPy* or *Matplotlib* from having to worry about each other's module names

# Importing Modules from Packages

```
import matplotlib.pyplot
```

- Loads the submodule `pyplot` from the package `matplotlib`

```
import numpy as np
```

- Loads all subroutines/functions from the package `numpy`
- Calling a specific function by using the dotted name convention, e.g. `y=np.sqrt(x)` for the square root function $y=x^2$

# Popular Packages (libraries)

https://hackr.io/blog/best-python-libraries

| | |
|---|---|
| • pandas, numpy, scipy …. | Data Handling/Analysis |
| • matplotlib, seaborn, bokeh …. | Data Visualization |
| • scikit-learn, tensorflow, pytorch, keras…. | Machine Learning |
| • beautifulsoup, scrapy …. | Web Scraping |
| • opencv, pillow …. | Computer Vision |
| • …. | |
| • …. | |

# Plotting in Python

- Before creating plots, it is worth spending sometime familiarising ourselves with a famous Python plotting library/package → **matplotlib**
- **`matplotlib`** was originally developed by a neurobiologist in order to emulate aspects of the MATLAB software
- **`matplotlib`** is an open-source Python library often touted as an alternative to the paid solution MATLAB. **`matplotlib`** was made for the purpose **of data visualization** as it's used to create graphs and plots.
- **`matplotlib`** does have a limit — it can only do 2D plotting. Despite this fact, this library remains highly capable of producing publish-ready data visualizations in the form of plots, diagrams, histograms, plots, scatter plots, error charts, and of course, bar charts.

# Different Graph Types

- A **simple line graph** can be plotted with `plot()`
- A **histogram** can be created with `hist()`
- A **bar chart** can be created with `bar()`
- A **pie chart** can be created with `pie()`
- A **scatter plot** can be created with `scatter()`

plus many more….

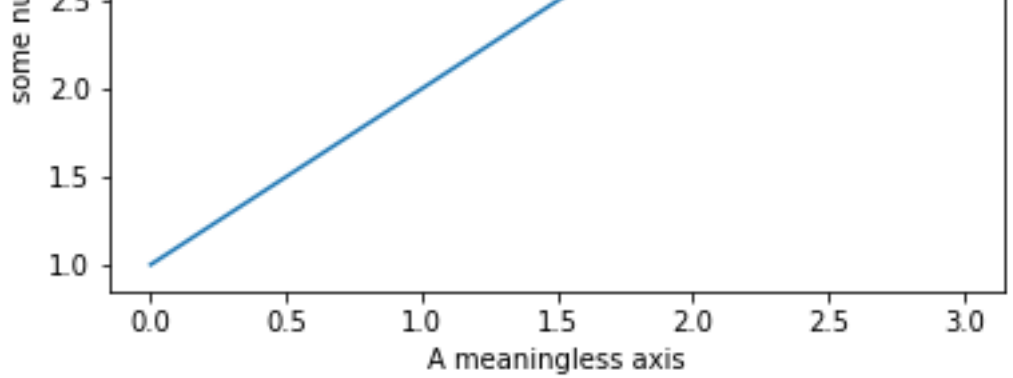
# Getting started

- We are also going to import **`numpy`**, which we are going to use to *generate random data* for our examples

```
import matplotlib.pyplot as plt
import numpy as np
```

# Our first plot

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.xlabel('A meaningless axis')
plt.show()
```




You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4.

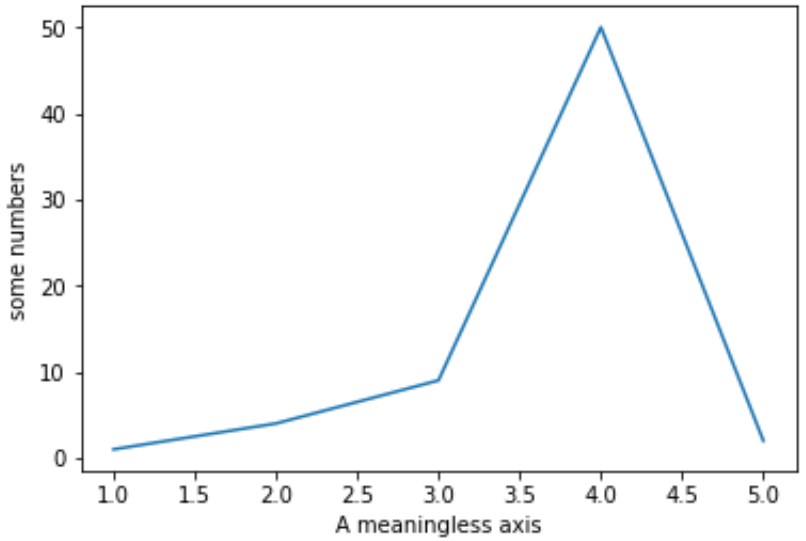
- If you provide a single list or array to the plot() command, Matplotlib assumes it is a sequence of y values, and automatically generates the x values for you.
- Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are [0,1,2,3].

# The `plot()` function

- The `plot()` argument is quite versatile, and will take any arbitrary collection of numbers.

For example, if we add an extra entry to the x axis, and replace the last entry in the Y axis and add another entry:

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2])
plt.ylabel('some numbers')
plt.xlabel('A meaningless axis')
plt.show()
```

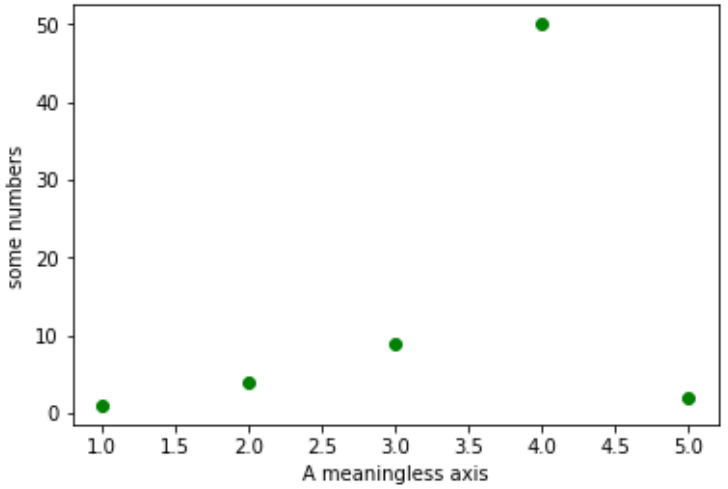


# The `plot()` function

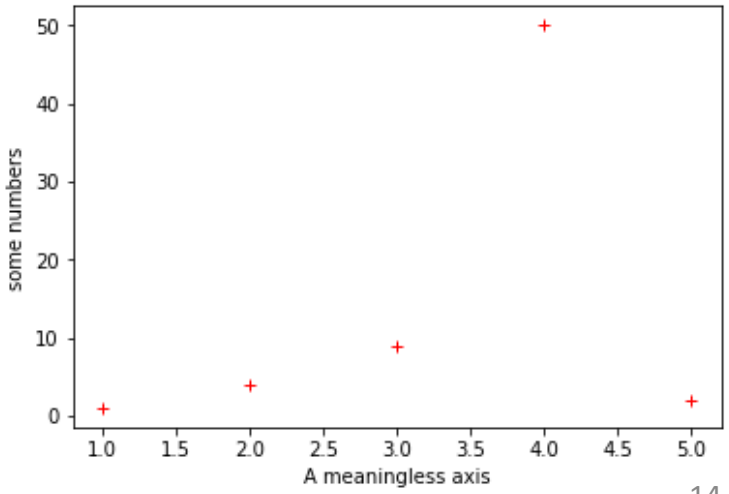
- The `plot()` function has an optional third argument that specifies *the appearance of the data points*.
- The default is `b-`, which is the blue solid line seen in the last two examples.

  The full list of styles can be found in the documentation for the plot() on the Matplotlib page

```
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], 'go')
```



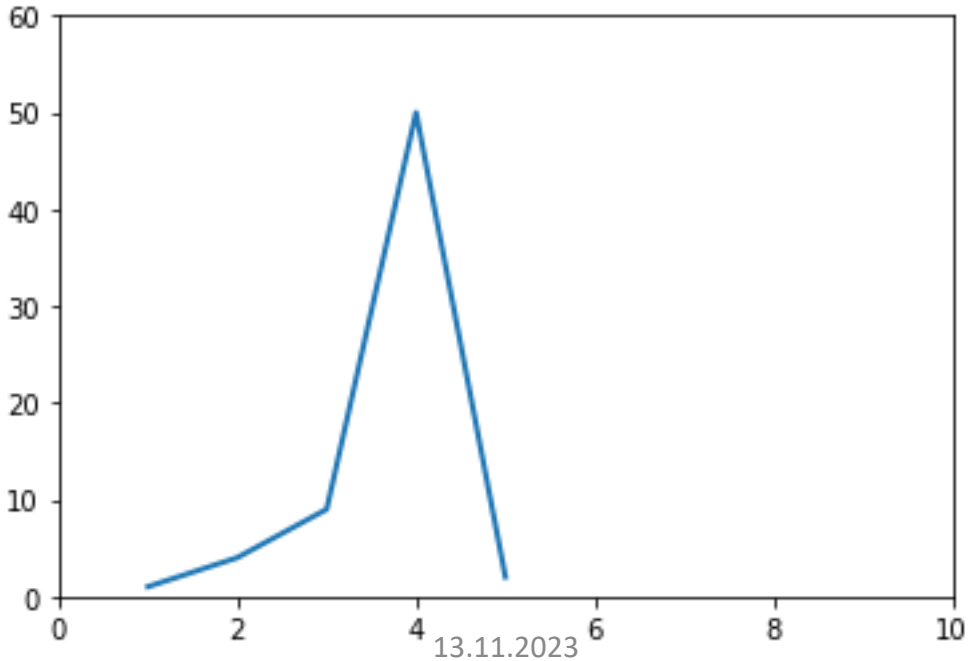

```
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], 'r+')
```



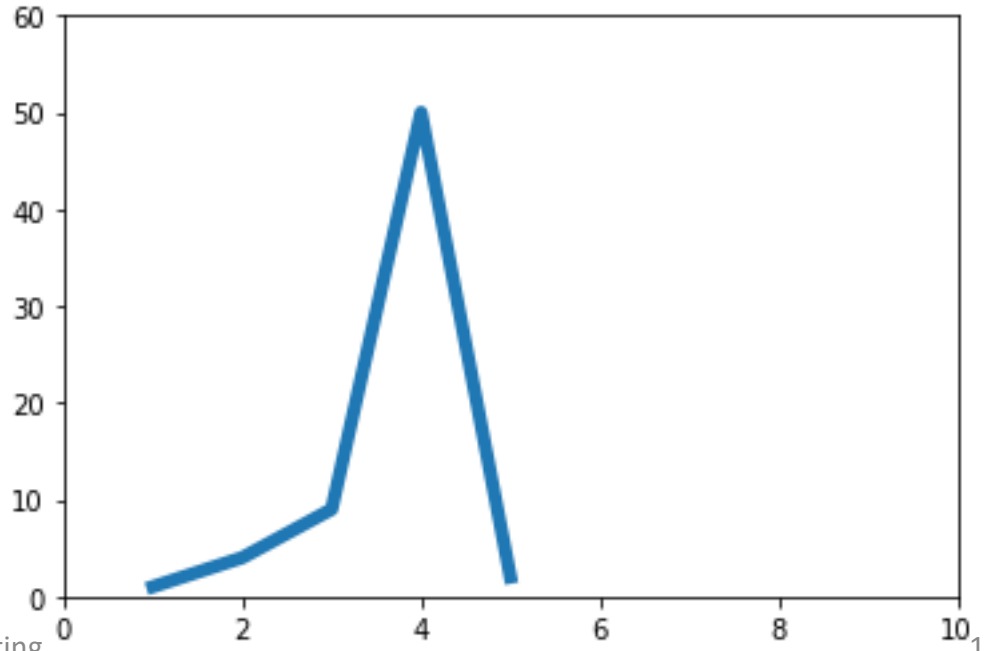
# The `plot()` function

You can quite easily alter the *properties of the line* with the `plot()` function.

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], '-', linewidth=2.0)
plt.axis([0, 10, 0, 60])
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], '-', linewidth=5.0)
plt.axis([0, 10, 0, 60])
plt.show()
```

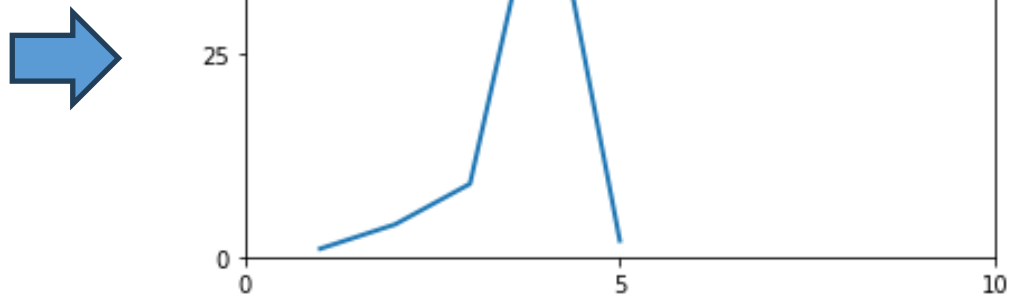


# Altering tick labels

- The `plt.xticks()` and `plt.yticks()` allows you to manually alter the ticks on the x-axis and y-axis respectively.

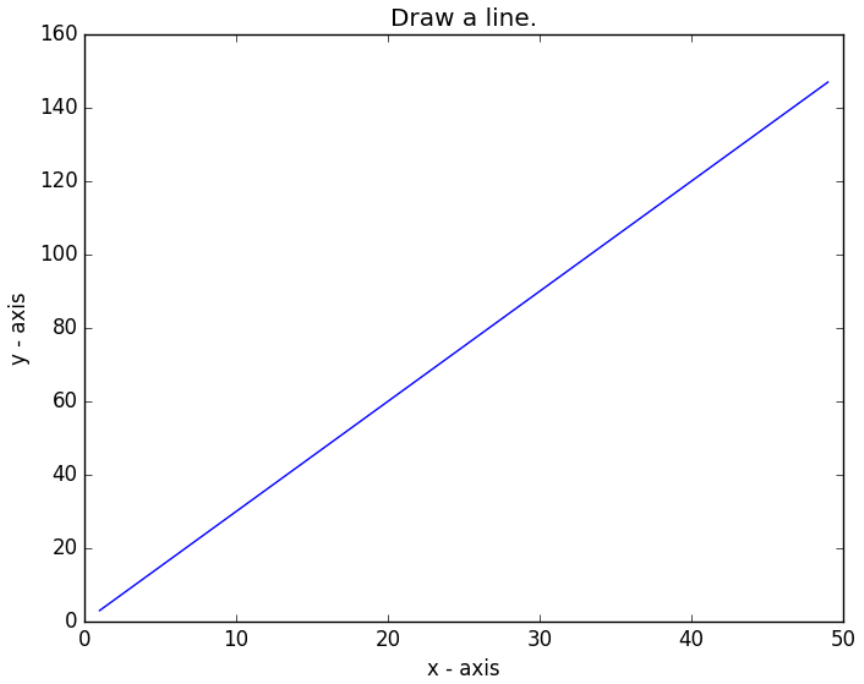
  Note that the tick values have to be contained within a list object.

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], '-', linewidth=2.0)
plt.axis([0, 10, 0, 60])
plt.xticks([0, 5, 10])
plt.yticks([0, 25, 50, 60])
plt.show()
```



# Task - Basic Line Graph

Let's write a Python program to draw a line graph with suitable labels for the x-axis and y-axis. Include a title.



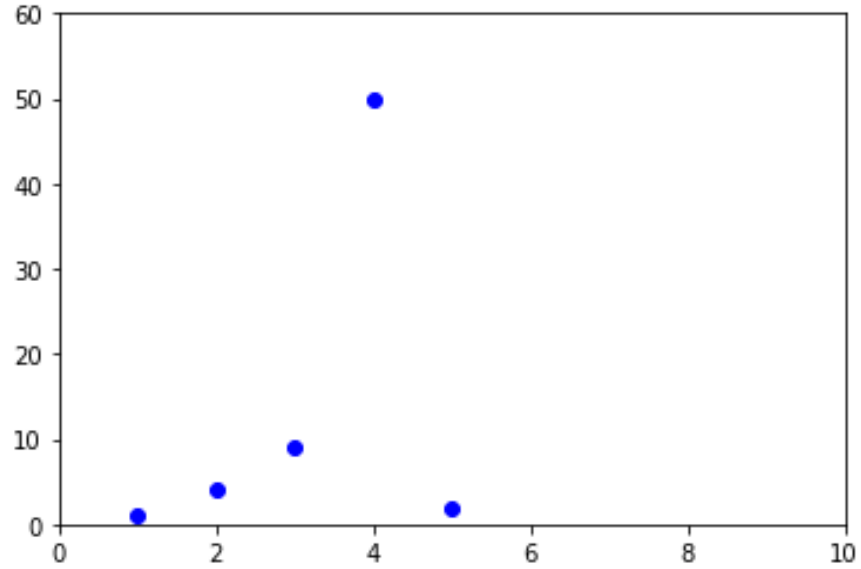

```python
import matplotlib.pyplot as plt

X = range(1, 50)
Y = [value * 3 for value in X]
print("Values of X:")
print(range(1,50))
print("Values of Y (thrice of X):")
print(Y)

# Plot lines and/or markers to the Axes.
plt.plot(X, Y)
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title
plt.title('Draw a line.')

# Display the figure.
plt.show()
```

# The `axis()` function

- The `axis()` function allows us to specify the range of the axis.
- It requires a list that contains the following:

  `[The min x-axis value, the max x-axis value, the min y-axis, the max y-axis value]`

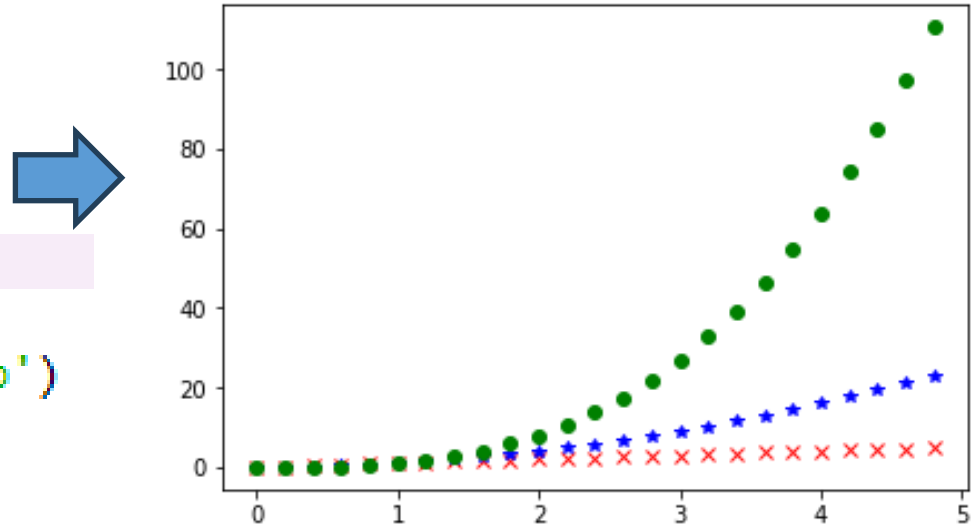
```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], 'bo')
plt.axis([0, 10, 0, 60])
plt.show()
```



# Matplotlib and NumPy Arrays

- Normally when working with numerical data, you'll be using **NumPy** arrays.
- This is still straight forward to do in **Matplotlib**; in fact all sequences are converted into NumPy arrays internally anyway.

```
import numpy as np
import matplotlib.pyplot as plt
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'rx', t, t**2, 'b*', t, t**3, 'go')
plt.show()
```



# Working with Text

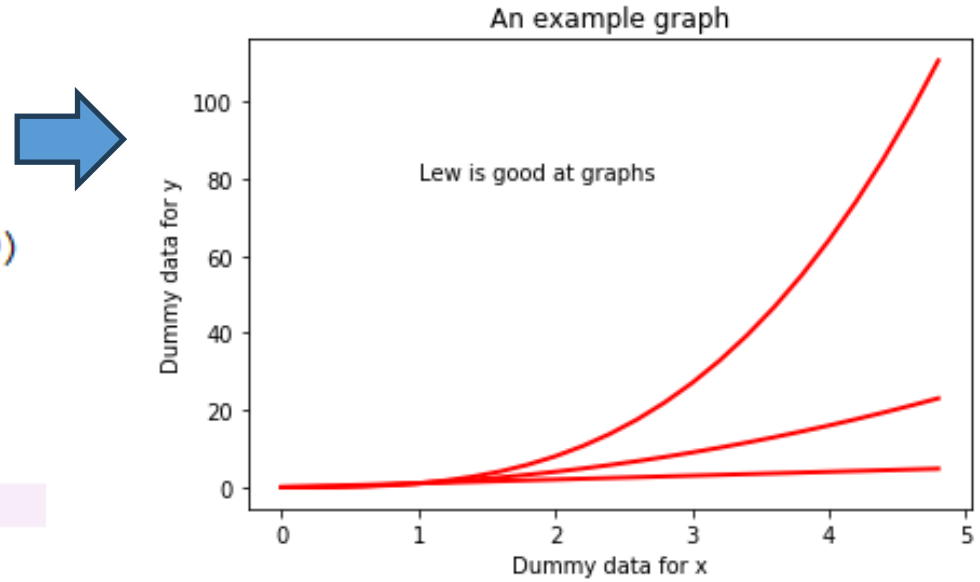
- There are a number of different ways in which to **add text to your graph**:
  - **`title()`** = Adds a title to your graph, takes a string as an argument
  - **`xlabel()`** = Add a title to the x-axis, also takes a string as an argument
  - **`ylabel()`** = same as **`xlabel()`**
  - **`text()`** = Can be used to add text to an arbitrary location on your graph. Requires the following arguments:

```
text(x-axis location, y-axis location, the string of text to be added)
```

- Note: Matplotlib uses TeX equation expressions. So, as an example, if you wanted to put $\sigma_i = 15$ in one of the text blocks, you would write `plt.title(r'$\sigma_i=15$')`

# Example - Working with Text

```
import numpy as np
import matplotlib.pyplot as plt
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
lines = plt.plot(t, t, 'b-', t, t**2, 'r-', t, t**3, 'g-', linewidth=2.0)
plt.setp(lines, color='r', linewidth=2.0)
plt.xlabel('Dummay data for x')
plt.ylabel('Dummy data for y')
plt.title('An example graph')
plt.text(1, 80, 'Lew is good at graphs')
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
plt.show()
```



# Legends

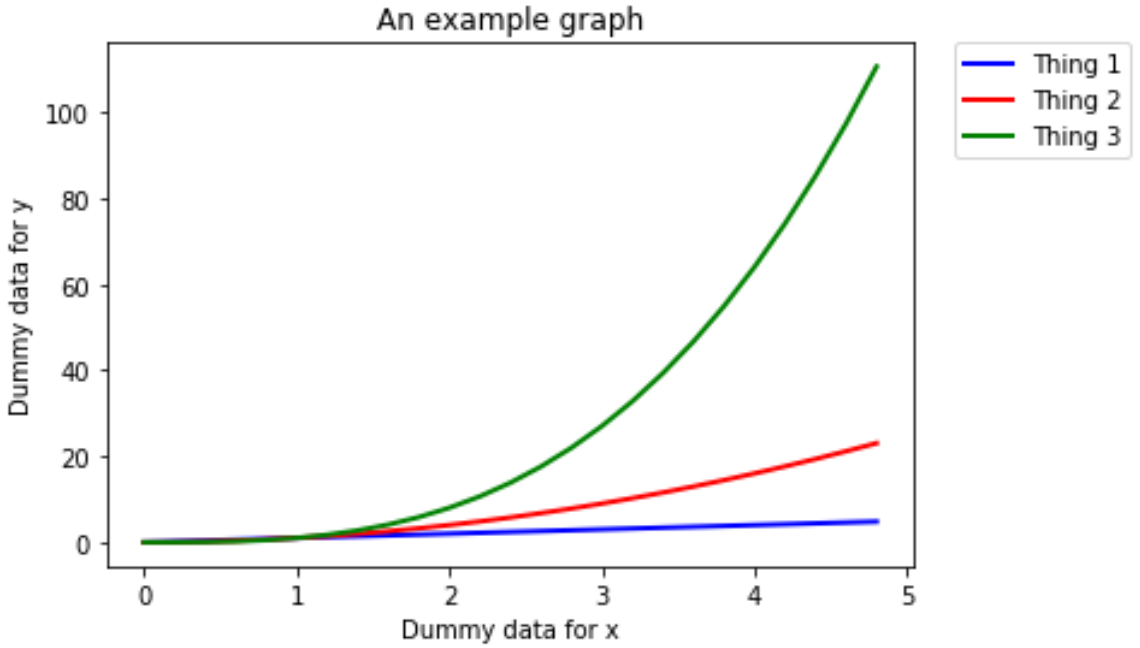
- The location of a legend is specified by the `loc` command.

  There are a number of in-built locations that can be altered by replacing the number. The Matplotlib website has a list of all locations in the documentation page for `location()`.

- You can then use the `bbox_to_anchor()` function to manually place the legend, or when used with `loc`, to make slight alterations to the placement.

# Legends

```
import numpy as np
import matplotlib.pyplot as plt
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
lines = plt.plot(t, t, 'b-', linewidth=2.0, label='Thing 1')
lines = plt.plot(t, t**2, 'r-', linewidth=2.0, label='Thing 2')
lines = plt.plot(t, t**3, 'g-', linewidth=2.0, label='Thing 3')
plt.xlabel('Dummy data for x')
plt.ylabel('Dummy data for y')
plt.title('An example graph')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



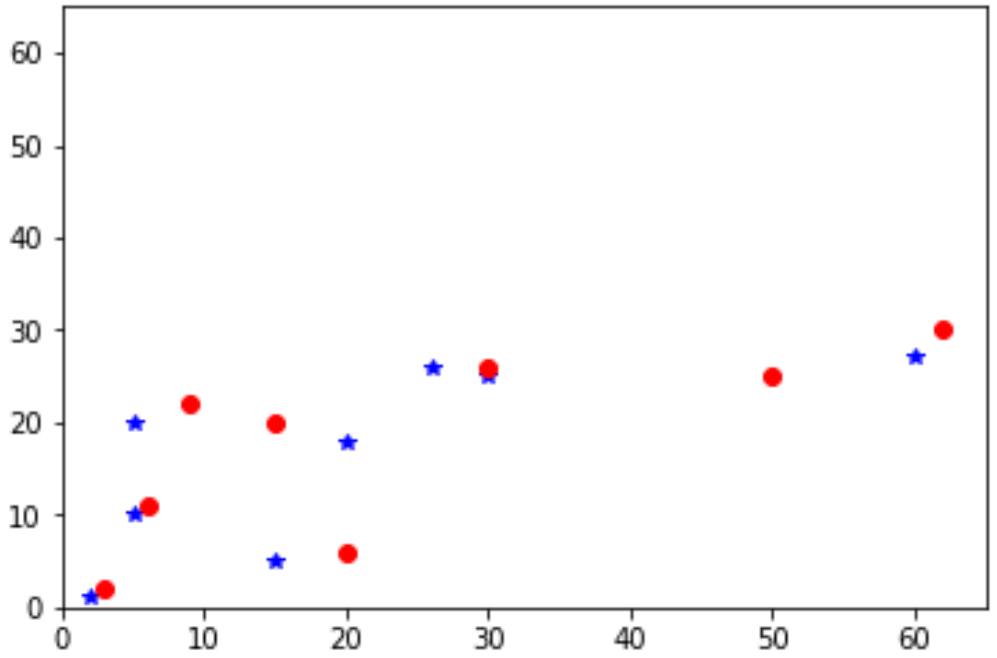
# Saving a Figure as a File

- The plt.savefig() allows you to save your plot as a file.
- It takes a string as an argument, which will be the name of the file. You must remember to state which file type you want the figure saved as; i.e. png or jpeg.
- Make sure you put the plt.savefig() before the plt.show() function. Otherwise, the file will be a blank file.

```
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
lines = plt.plot(t, t, 'b-', t, t**2, 'r-', t, t**3, 'g-', linewidth=2.0)
plt.setp(lines, color='r', linewidth=2.0)
plt.xlabel('Dummy data for x')
plt.ylabel('Dummy data for y')
plt.title('An example graph')
plt.text(1, 80, 'Lew is good at graphs')
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
plt.savefig('test.png')
plt.show()
```

# Task – Scatter Plot

Let’s write a Python program to plot quantities which have an x and y position; a scatter graph.




```
import numpy as np
import pylab as pl

# Make an array of x values
x1 = [2, 15, 5, 20, 5, 30, 26, 60]
# Make an array of y values for each x value
y1 = [1, 5, 10, 18, 20, 25, 26, 27]
# Make an array of x values
x2 = [3, 20, 6, 15, 9, 30, 50, 62]
# Make an array of y values for each x value
y2 = [2, 6, 11, 20, 22, 26, 25, 30]


# set new axes limits
pl.axis([0, 65, 0, 65])
# use pylab to plot x and y as red circles
pl.plot(x1, y1,'b*', x2, y2, 'ro')
# show the plot on the screen
pl.show()
```

repl.it