

LEARN PYTHON WITH EDUBLOCKS

This handout is focused on using Edublocks to learn the Python programming language

This document includes a review of the fundamentals of programming using EduBlock including:

- 1) Variables, input/output, and expressions
- 2) Decision Making
- 3) Iteration; - basic for and while loop, continue and pass
- 4) Arrays and Lists
- 5) Decomposition and functions
- 6) Relational operators, logical operators for compound conditions

In addition, more advanced programs using EduBlock and transition to python are covered in this tutorial.

Table of Contents

Chapter 1: Introduction: Getting Started	04
Chapter 2: The EduBlock Environment	09
Chapter 3: Part 1 - Basic Concepts of Python	10
Hello, World!	10
Simple Input Program	11
Simple declaring variables	12
Chapter 4: Simple Programs	13
Add two numbers - Without user inputs	13
Add two numbers - With user inputs	14
Subtract two numbers - Without user inputs	15
Subtract two numbers - With user inputs	16
Using addition and subtraction with the Turtle Library	17
Chapter 5: Conditional statements / Iteration	19
If..else	19
Example of if...else with turtle library	21
For Loop	22
Example of for loop with turtle library	22
While Loop	23
Example to find odd numbers using while loop	23
Example of while loop with turtle library	23
Continue Statement	25
Pass Statement	26
If...else & for loop Combination (Prime Number example)	27
Chapter 6: Arrays and Lists	28
List - Declaration and Printing	28
Example of lists with turtle library	29
List - Appending to a list	30
Chapter 7: Part 2 - Advanced Concepts	32
Functions, definitions, and other concepts	32
How to run the Sample Examples	33
Simple Calculator	35
Temperature Converter	37
Roots of a quadratic Equation	39
Simple Pie Chart showing world population on different continents	41

Chapter 1

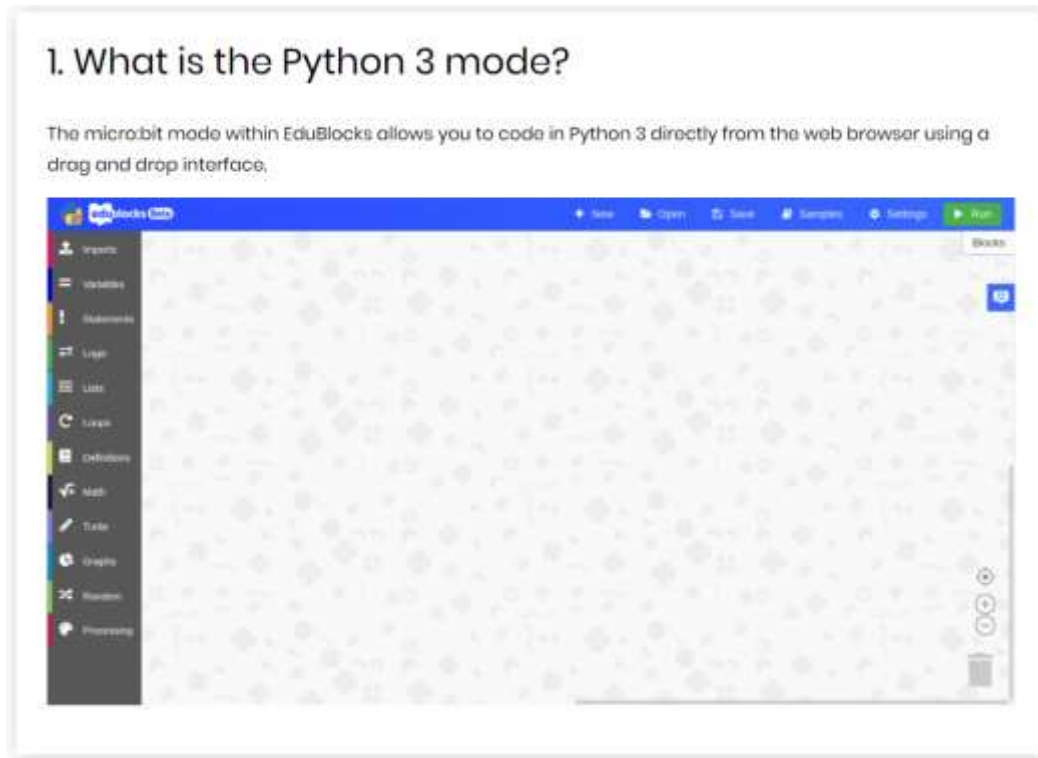
Introduction - Getting Started

1. Go to the website <https://learn.edublocks.org/>
2. Scroll down and select “Getting Started with Python”



3. Follow the 5 steps instructions on Getting started with Python using Edublocks which shows:

- How to get started - <https://app.edublocks.org/>



- What is Python 3 and how Edublocks help to understand the programming language and use it to learn coding in Python language

2. What is Python 3

Python is a powerful multi-purpose programming language created by Guido van Rossum. It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

The EduBlocks Python 3 mode offers:

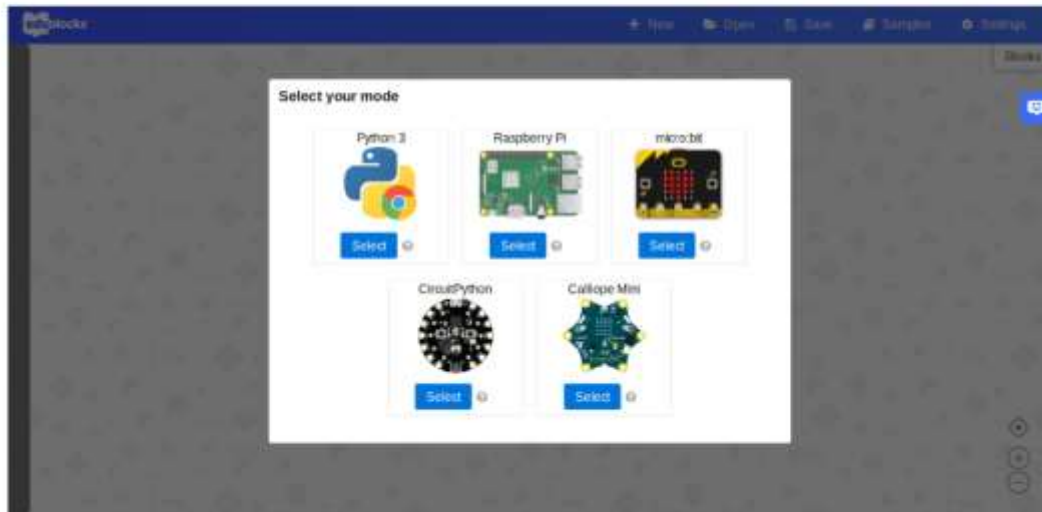
- Text on the blocks: Python code is displayed on the blocks so as you're coding in a familiar block environment, you're learning the Python syntax,
- Install free: The Python 3 mode is web based, meaning it's completely install free. All you need is an internet connection to start coding. It's cross-platform too!
- Teaches core python: The Python 3 mode covers all the core python syntax/concepts helping you feel more comfortable when coding the real thing.
- Turtle Support: Code Python Turtle with EduBlocks on the web. Draw shapes, create cool graphics and more using python code and the turtle library!
- Python text view: Create your Python code in the block editor, and with a click, you can switch to a Python text editor. You can also download the .py file too!
- Create Graphs: Python is known for it's use within data science. The Python 3 mode has support for PyGal, a python graph creation library allowing you to easily create graphs.

- Once you've selected the Python mode, you should see it pop up

3. Launching the Python 3 mode

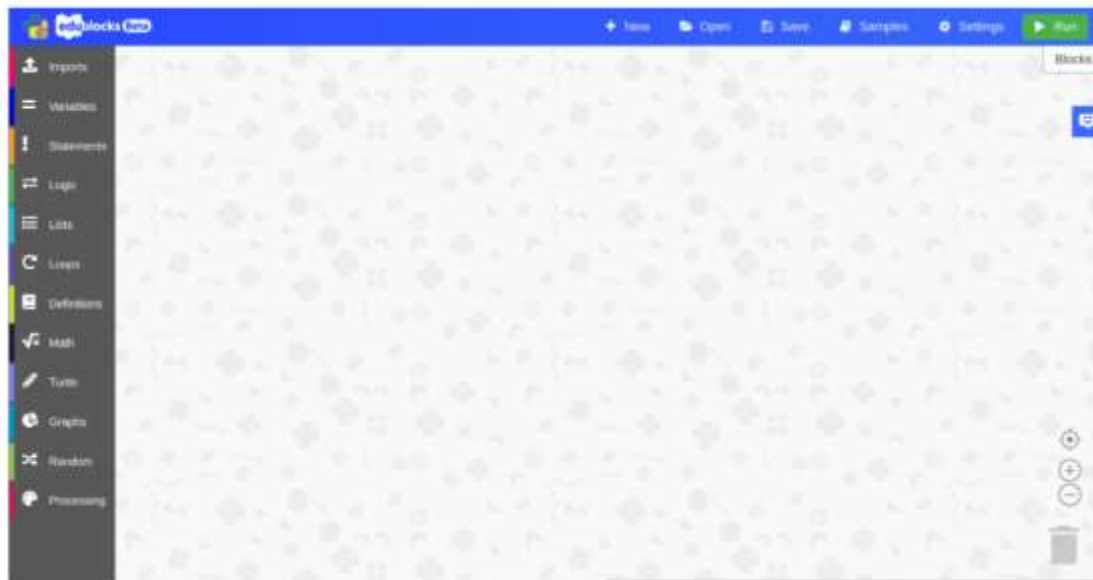
So, you want to get started with EduBlocks and Python 3?

You can do this by opening a web browser of your choice and typing <https://app.edublocks.org> into the search box. Once you've loaded up EduBlocks, you'll be presented with the mode selector.



Now, we want to select the Python 3 mode. To do this simply click on the blue select button underneath the Python icon. This will load up the Python mode.

Once you've selected the Python mode, you should see it pop up:

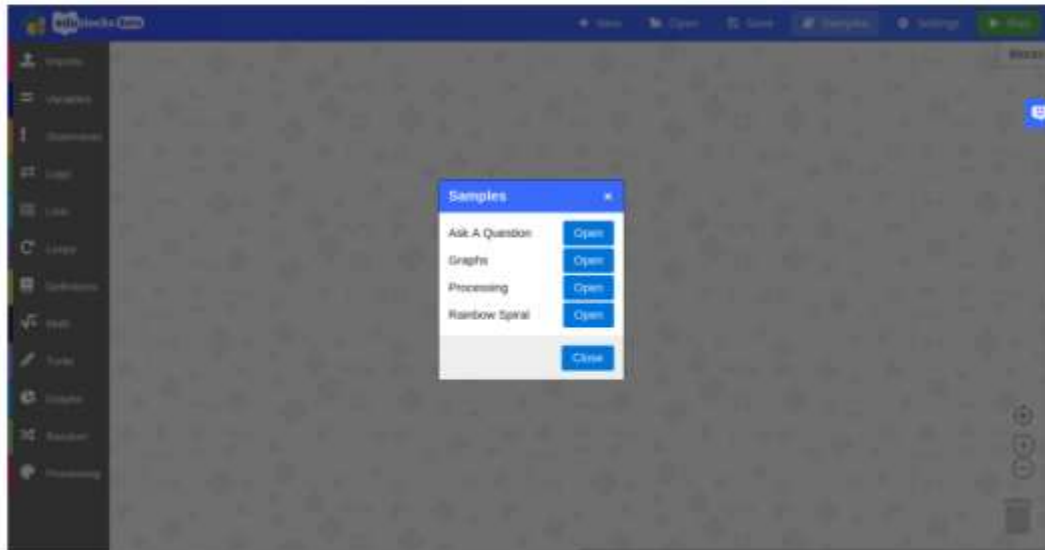


- Try to load up the sample to get an idea of the functioning

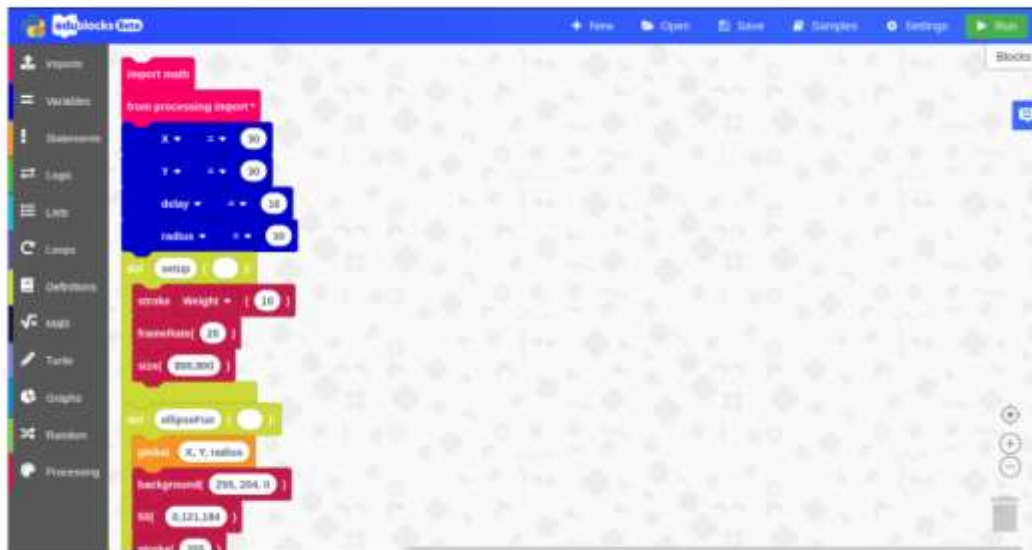
4. Loading up samples

Once inside the Python 3 mode, we have a range of different things we can do. Before you write your own program it's a good idea to load up a sample so you can see how EduBlocks works.

To load up a sample, click the samples button in the blue navbar at the top, this will load up the following dialog box:



Click on the blue open button next to the "Processing" sample.
Once you've done this, a few blocks should appear in the workspace.

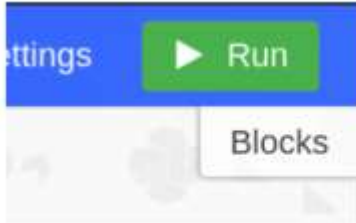


- Execute the code and get the output

5. Run Python 3 code

Here is how to run code in the Python 3 mode:

Once you've completed your program, you can press the green run button in the top right hand corner to run your code.



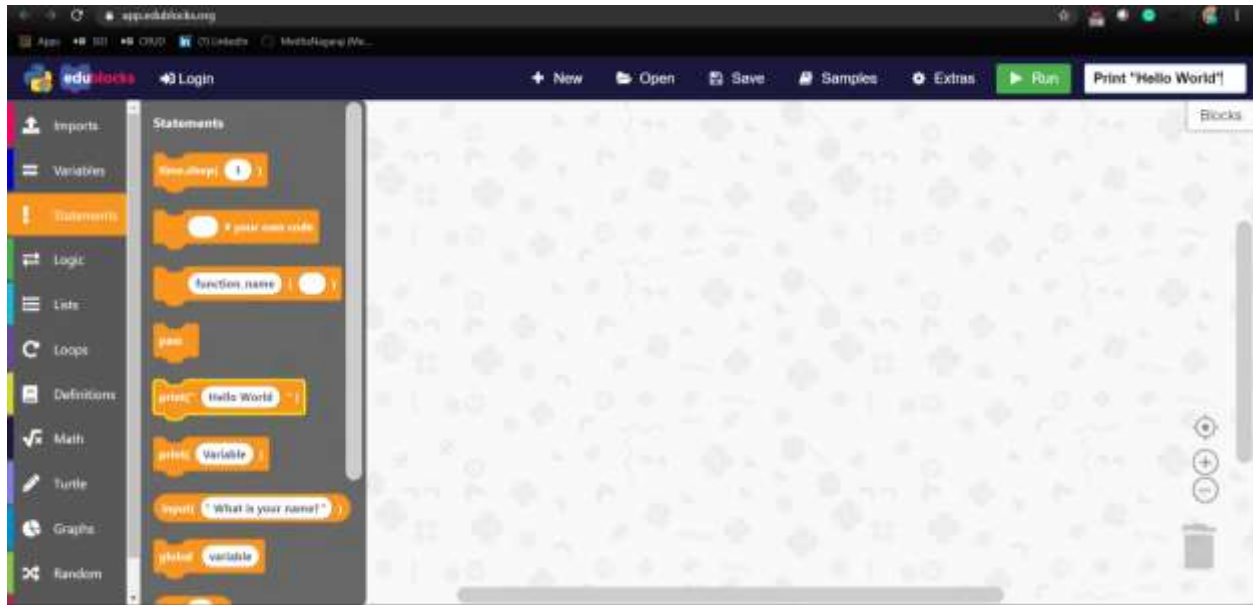
This will load up a Python shell window in the browser where you'll see the output of your code.

Chapter 2

The EduBlock Environment

This chapter gives you a brief introduction to the EduBlock Environment

Below is a screenshot of how the environment looks like. Let's get started **with** the basics of python programming language and simple coding examples.



Chapter 3

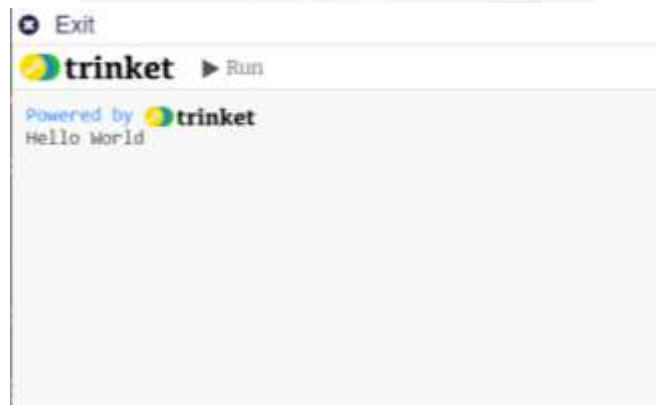
Part 1 - Basic Concepts of Python

Part 1: Basic Python Programming Concepts

-> Hello, World!

Let's start with the **Console/Terminal**. In that section, let's create a block to print "hello world":

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

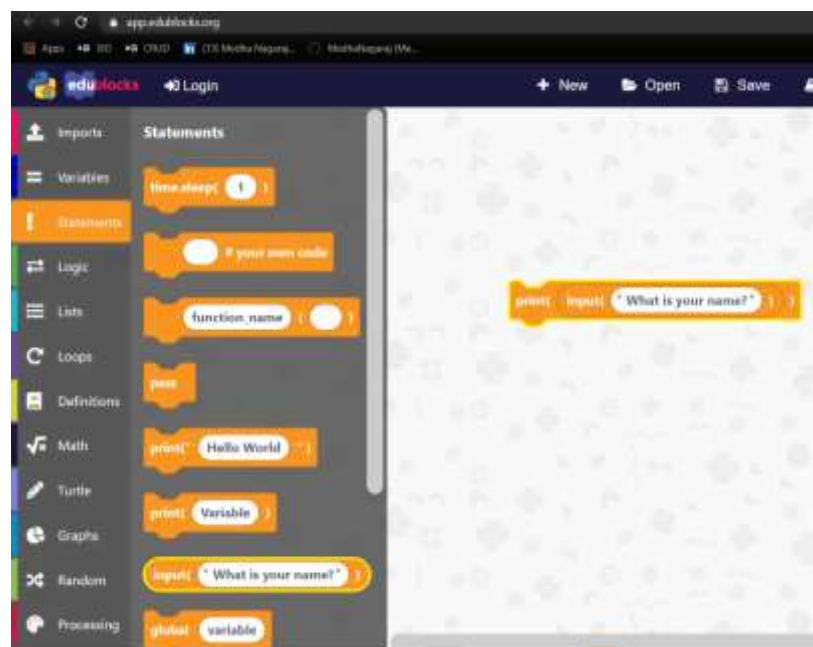


-> A Simple Input Program

Now let's move over to the **Editor**. Type in the following python code:

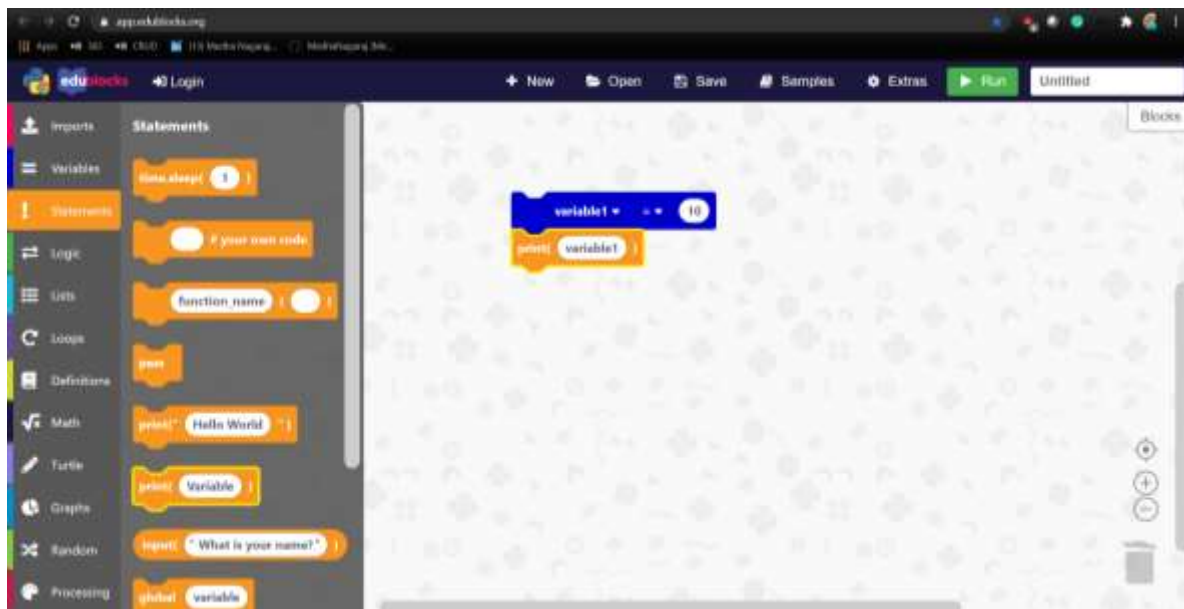
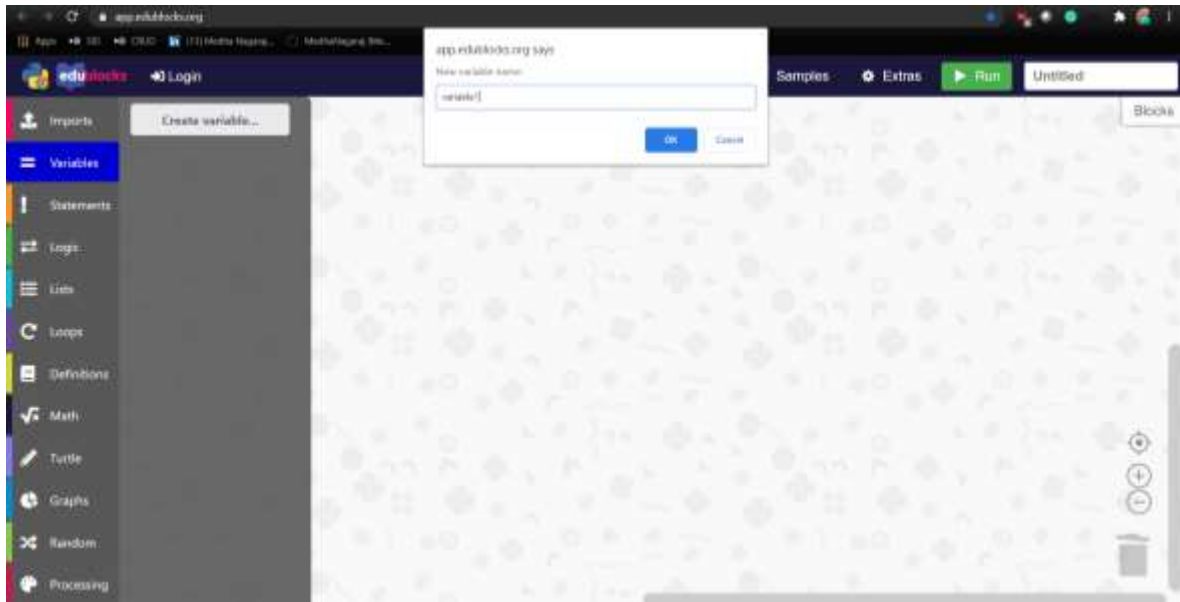
```
print(input("What is your name?"))
```

To run it, click on the run button on the window.



-> A Simple Program - declaring variables

Select the 'Variables' tab from the menu and click on the 'Create variable' button and enter a suitable name for the variable.



✕ Exit

 **trinket** ▶ Run

Powered by  **trinket**
10

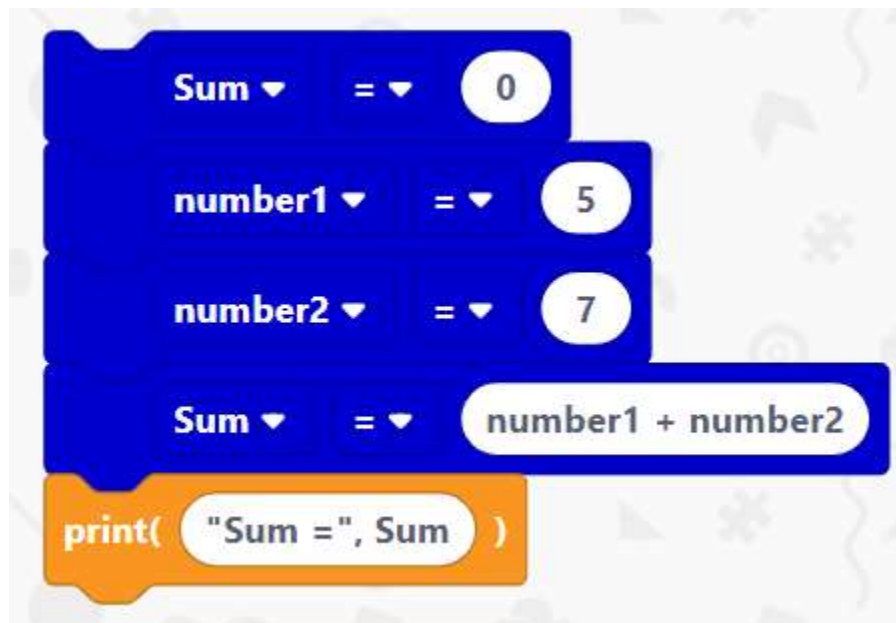
Simple Programs

-> Program to add two numbers - Without user inputs

In programming, when we try to calculate something we have to form an expression which when it includes arithmetic operators, it is normally called an arithmetic expression. For example, $a = b + c$

Notice that in programming, we use the equal sign (=) to transfer the result of the expression into the variable a .

In the following, we show you how to add number1 and number2 from each other and show the results in variable *addition*. The numbers are already declared in the program beforehand.

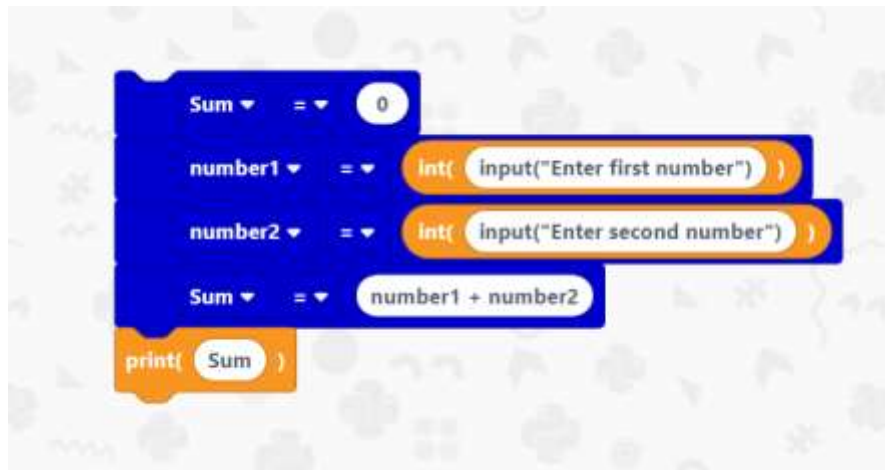


-> Program to add two numbers - With user inputs

In this example, we are demonstrating the addition of two numbers. For this purpose we will be declaring two variables (numbers), *number1* and *number2* that are added together using an arithmetic expression and are defined as another variable '*sum*' such as :

$$sum = number1 + number2$$

We are taking inputs from the user, i.e., the user will be entering the numbers of their choice to perform the addition operation. Here we use the *input* function to take the input and make sure to convert it to *int* type.

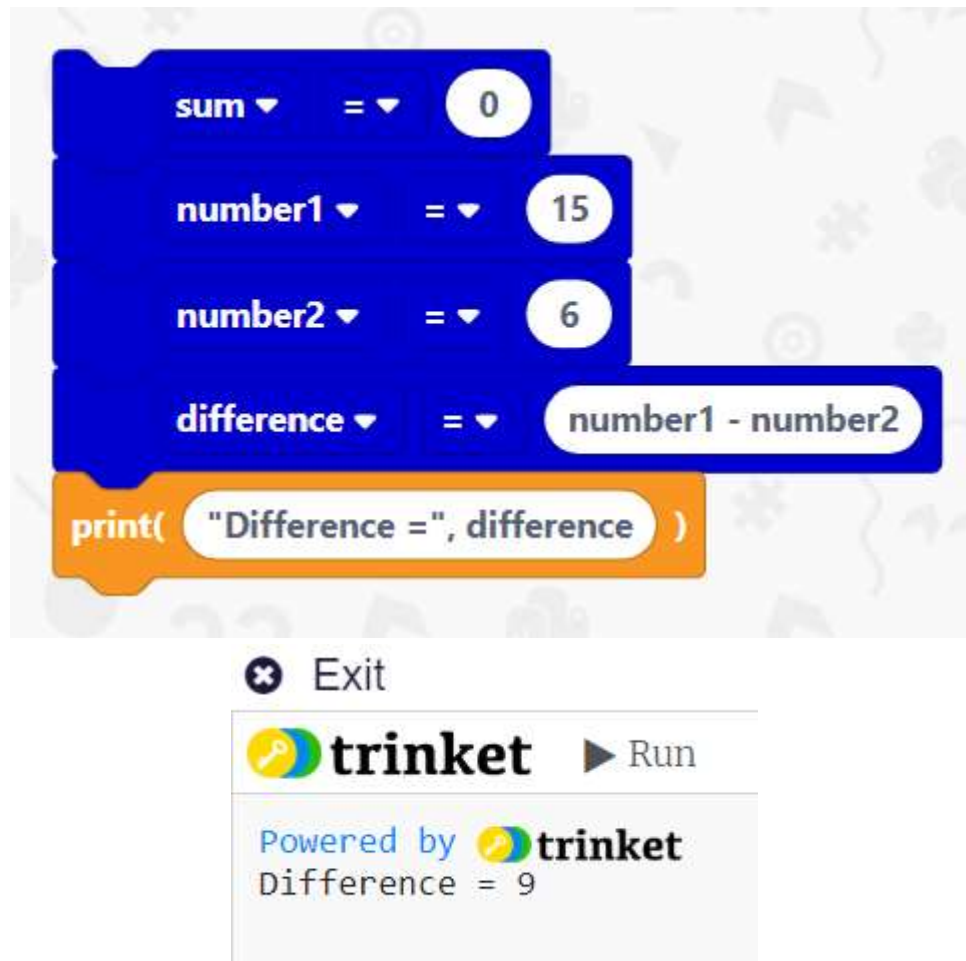


-> Program to subtract two numbers - Without user inputs

Here we will be replicating the arithmetic expression, $a = b - c$

Notice that in programming, we use the equal sign (=) to transfer the result of the expression into the variable a .

In the following, we show you how to subtract number1 and number2 from each other and show the results in variable *difference*.



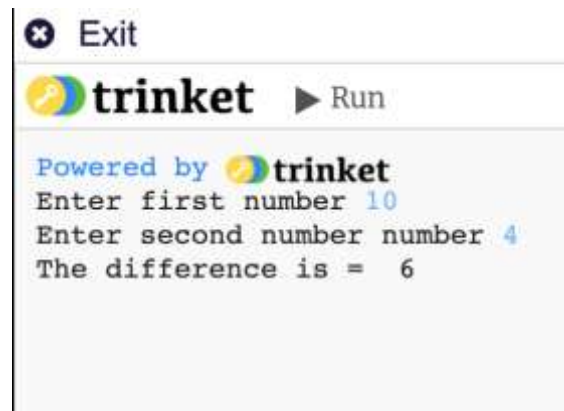
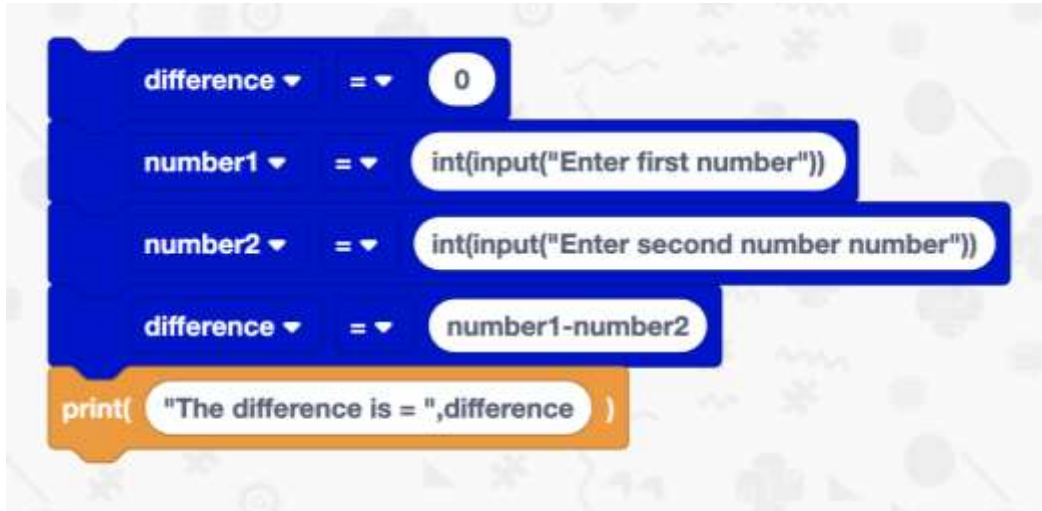
Try it yourself

1. Addition of two numbers
2. Subtraction of two or more numbers
3. Take a user input and do addition and subtraction

-> Program to subtract two numbers - With user inputs

In this program, we are accepting two numbers as inputs from the user using the *input* keyword and converting them to *int* type.

Notice that in programming, we use the equal sign (=) to transfer the result of the expression into the variable *difference*.



Try it yourself

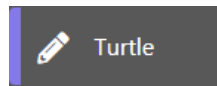
1. Multiplication of two numbers
2. Division of two numbers. Also handle case where the divisor is 0

-> Using addition and subtraction with the Turtle Library

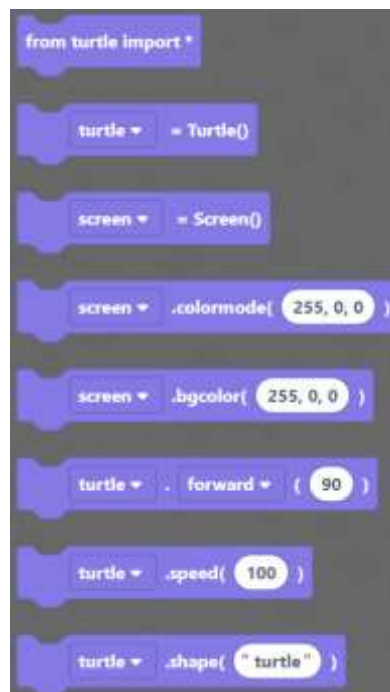
Python has a built-in turtle library primarily used for learning object-oriented programming and graphics.

A more in-depth explanation of the library is available [here](#). As before, we start from EduBlocks.

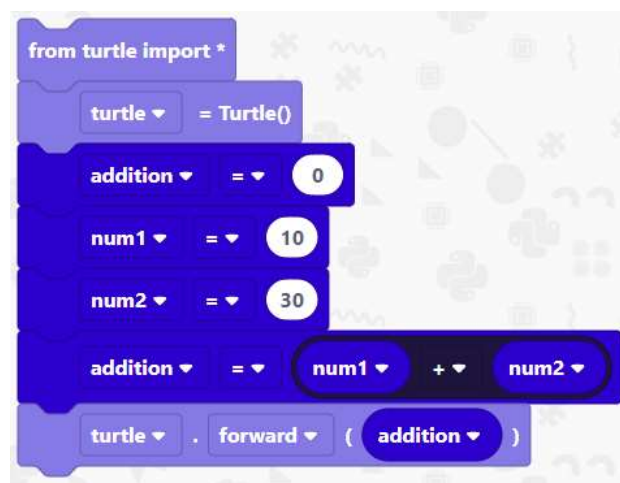
In EduBlocks, there is a block category dedicated solely to the use of the Turtle objects. In the EduBlocks interface, you will find the Turtle library on the left sidebar. It should look like this:



When clicking on the library, you will find all the different block statements you can use.



Here is an example of how we can implement addition and subtraction using the Turtle library:



Output will look like this:



If you look at the above example, we are doing the same type of arithmetic operation by adding two numbers, *num1* and *num2*. However, we use the sum of these two numbers to move the turtle forward the total number of pixels which resulted in the addition.

Block in EduBlocks	What it does
	Imports the turtle library to your program
	Creates a new Turtle object called "turtle" which can then use the functionality of the library.
	Moves the turtle forward 90 pixels

****Try it yourself****

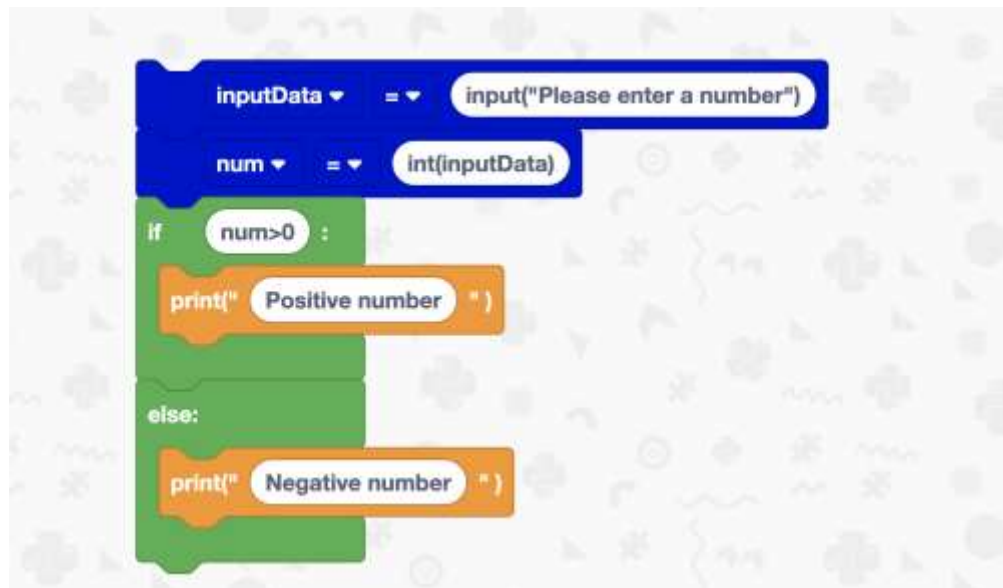
Create a similar program as the example above which moves a Turtle object forward; however, instead of predefining the values of *num1* and *num2*, allow the user to input the two values to add. You can use the previous examples of user input to help with this.

Conditional Statements

->If...else statement

In programming, we have a very important construct called decision making. All programming languages have conditional statements to perform decision making. A common conditional statement in almost every language is called *if ... else* statement. Basically, in the if part of the statement, a condition is checked or a logical expression is evaluated resulting in true or false. If the result is true, the statements inside the if-block will be executed while if it is false, the statements inside the else-block will be executed.

Example: Check if an input number is a positive or a negative integer. In this example, an integer called *num* is entered by the user and then is checked to see whether it is true or false as shown. Notice that if your integer is a 0, the program will have a problem. How do you fix it?



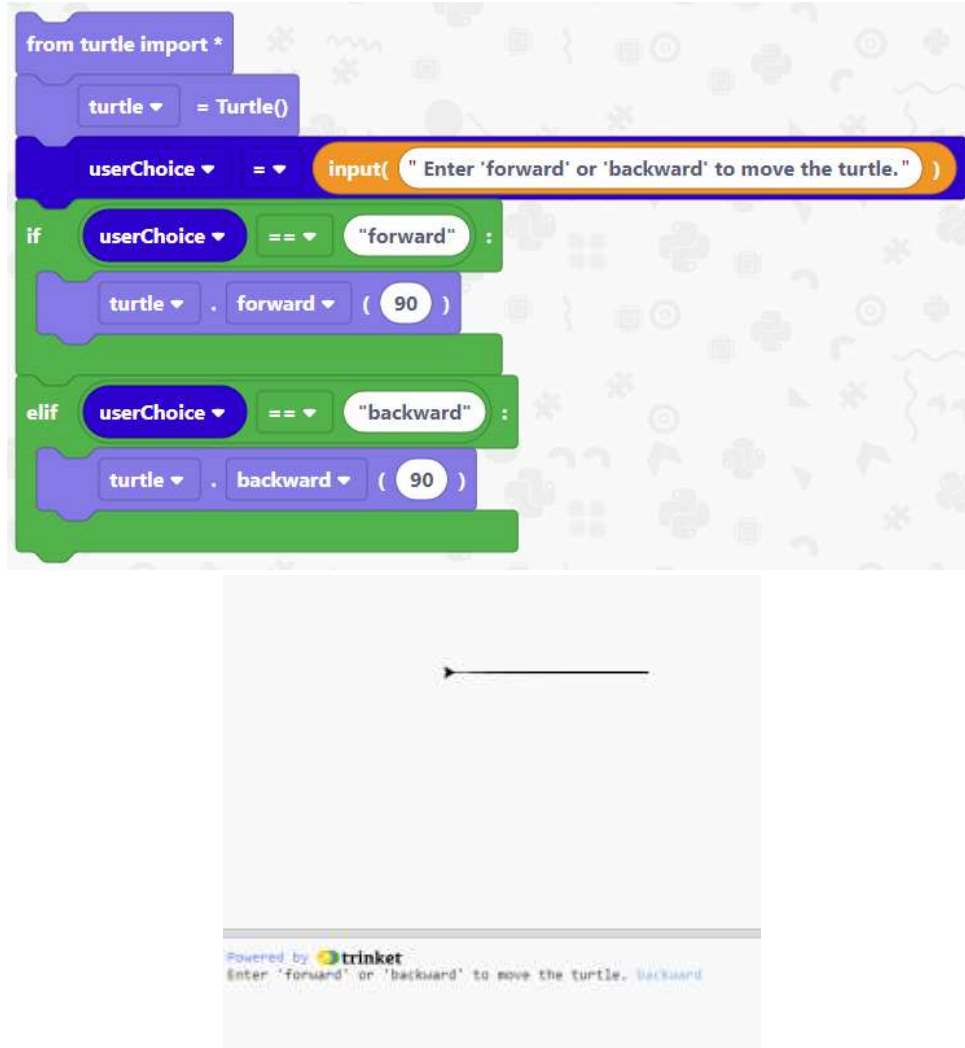
Now, try to see how by adding an elif-block with if...else, we could take off 0 as well so that an integer can be classified as Positive, Negative, and Zero.



*****Try it yourself*****

1. Write a program to take input as age of a person, and check whether the person is of legal voting age or not (18+ years)
2. Write a program to provide grades to students according to their scores.
Take input as *score* in *int* format. Grades to be provided are as below
score \geq 90 - A
score \geq 80 - B
score \geq 70 - C
score \geq 60 - D
Score $<$ 60 - F

-> Example of if...else with Turtle Library:



In this example, we are asking the user to enter either “forward” or “backward”, and depending on the input, we move the turtle in the specified direction 90 pixels. We use an *if* and *elif* statements to check the user input.

****Try it yourself****

Add to the above example so that the user can also enter “right” or “left”, and the turtle will move in those directions. You will need to use the **turtle.right()** and **turtle.left()** blocks to turn the turtle **90 degrees either direction before moving**. You can find these blocks by first dragging the turtle.forward() block and clicking the dropdown:



-> For loop

Another main construct in programming is called “Iteration” or “loops”. There are several standard loops known as for-loop, while-loop, and do-while loop. In this part, we explain the for-loop.

A for-loop is a type of iterator in programming that loops a specified number of times based on the condition. In the below example, the for loop is defined as “*for num in list1*”;, meaning it will loop through every element in the list named “*list1*”. For the first iteration, it will begin at the first element located at index 0. The value at index 0 is 10, so **num** will store **10** for the first loop. It will run all of the code inside of the for-loop, and once it is complete, it will move on to the second element. This will keep looping until it has reached the last element of *list1*.

Example: Find odd numbers from a list



Note: The condition “ $num \% 2 \neq 0$ ” contains the modulus $\%$ operator. This returns the remainder of the division of the two values. For example, if *num* stored the value 10, the expression $num \% 2$ would return 0, as 10 divided by 2 would return no remainders.

Example of for loop with Turtle Library:

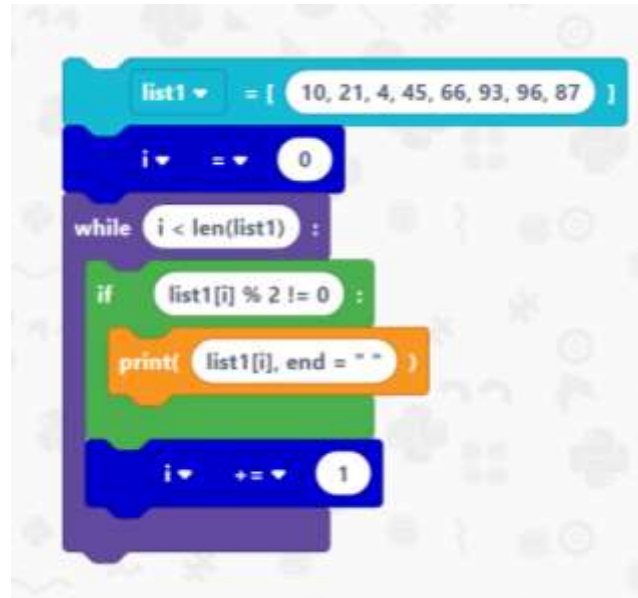


In the above example, we use a for-loop which iterates 10 times. For each iteration, the turtle moves forward 10 pixels. In total, the turtle would move 100 pixels.

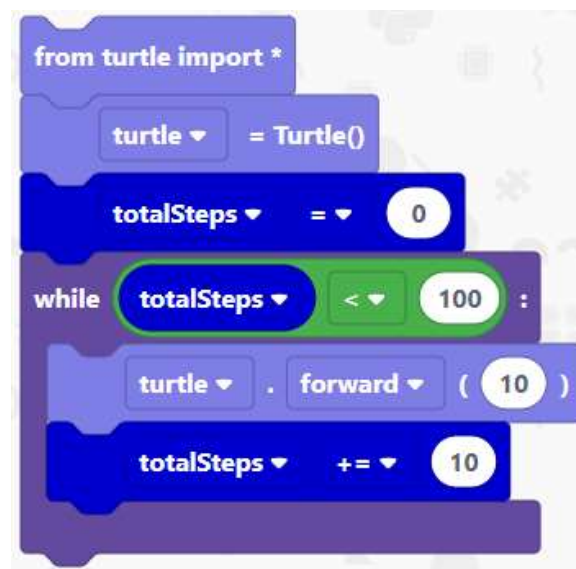
-> While loop

A while loop is another type of iterator which will *keep looping until the condition is false*. In the below example, the while loop is defined as: *while i < len(list1)*. This means that the code inside of the loop will continue to repeat until the variable 'i' becomes equal to or greater than the length of list1. Looking inside of the while loop, you will see that every time a loop is complete, 'i' will increment by 1. Therefore, 'i' will start at 0, and once it has looped 7 times, the while-loop will exit.

Example to find odd numbers from a list using *while loop*:



Example of *while loop* with Turtle Library:



In this example, we use a while loop to once again move the turtle a total of 100 pixels. Instead of directly defining how many times the loop will iterate such as in the for-loop, we create a condition.

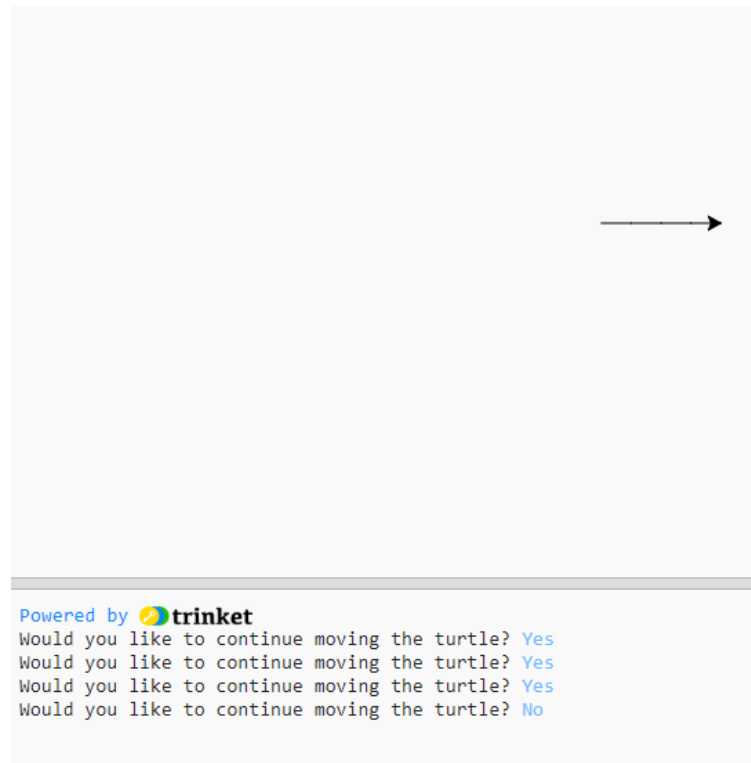
This while loop will continue to iterate until the condition becomes **false**. Therefore, since our condition is *totalSteps* < 100, the turtle will continue to move 10 pixels at a time until it has reached 100 pixels total (with *totalSteps* as the variable to keep track of this). When the condition becomes false, the loop will exit.

****Try it yourself****

Using the previous example as a reference, now create a program so that the turtle movement will continue to loop until the user enters “No” when asked if they would like to keep moving the turtle. For every iteration, it should first move the turtle forward 10 pixels, then ask the user for input.

In order to do this, you will need to create a variable to store the user’s input. Inside of your while loop, have it prompt the user for a choice. For your while loop’s condition, you will need to use the **!= operator** (does not equal) to check the user’s choice.

Example output:



-> Continue statement

The *continue* statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. In this example, when the num gets updated to 3 we are going to continue the loop instead of printing the value. The *continue* statement makes the loop continue to the next iteration



✕ Exit

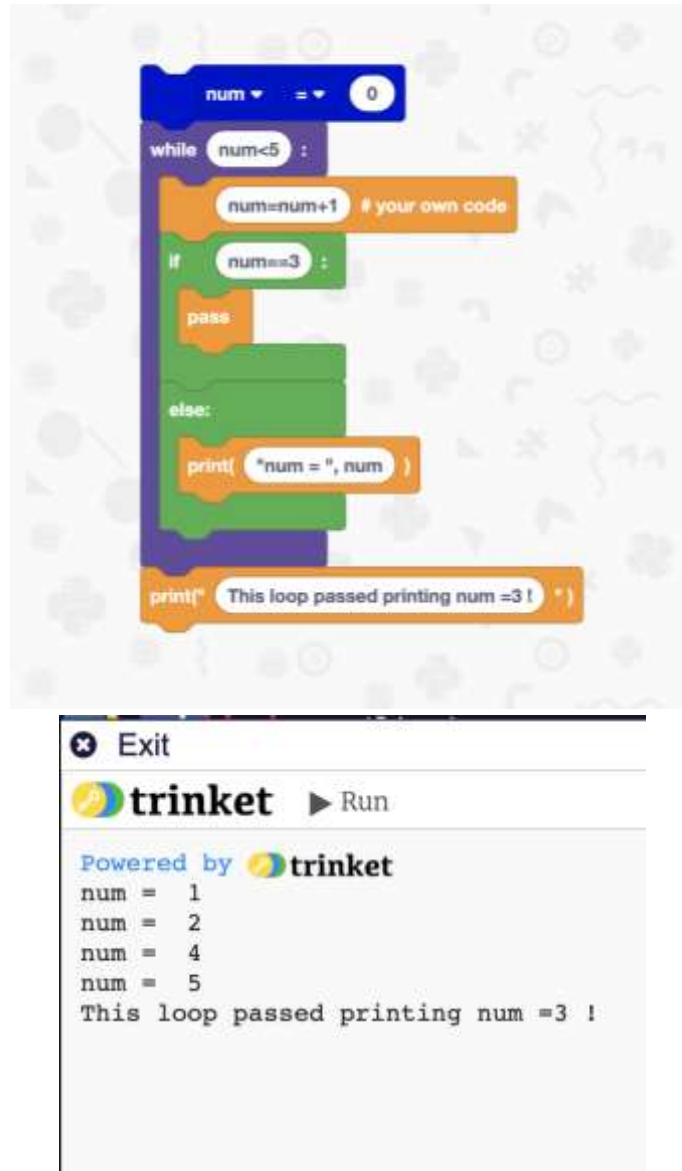
 **trinket** ▶ Run

Powered by  trinket

```
num = 1
num = 2
num = 4
num = 5
This loop skipped printing num =3 !
```


-> **"pass" statement**

The *pass* statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. This is a null operation -- when it is executed, nothing happens.



*****Try it yourself*****

1. Print all the multiples of 3 between 1 and 100 using for both for-loop and while-loop

Note:

Use `%` (modulus) operator - Modulus operator works on integers (and integer expressions) and yields the remainder when the first operand is divided by the second

if(`number % 3 == 0`) - then number is a multiple of 3

-> Combination of If...else and for-loop

Example: Find if a given number is prime or not

Definition: A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. The first few prime numbers are {2, 3, 5, 7, 11,}.

Examples :

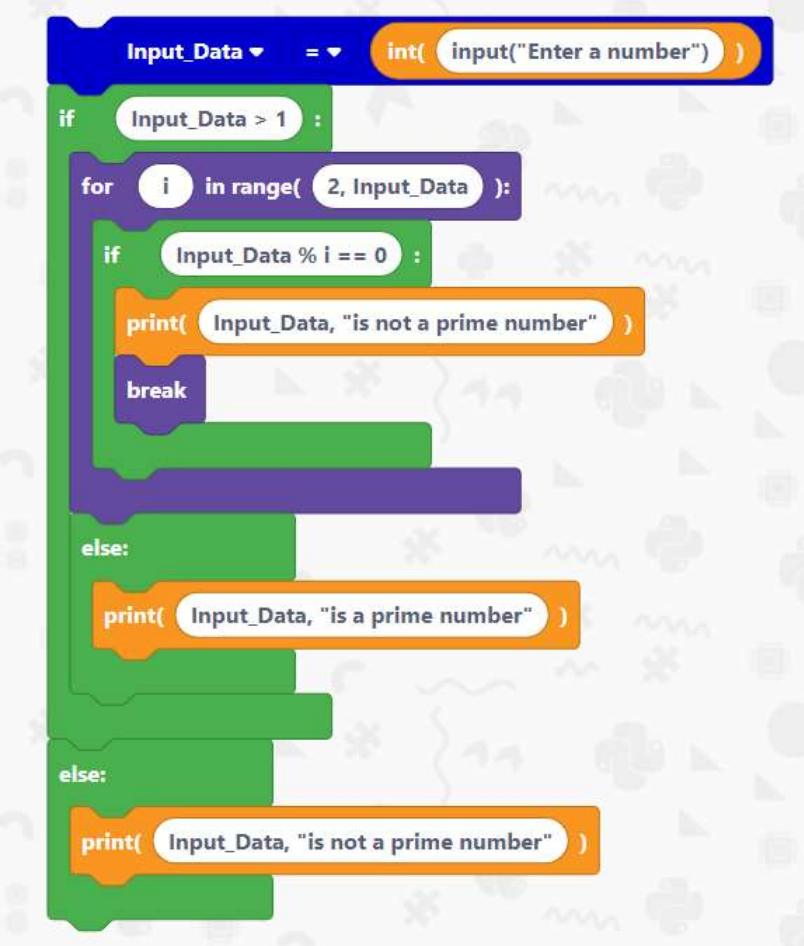
Input: $n = 11$

Output: true

Input: $n = 15$

Output: false

The idea to solve this problem is to iterate through all the numbers starting from 2 to $(n/2)$ using a *for-loop* and for every number check if it divides n . If we find any number that divides, we return false. If we did not find any number between 2 and $n/2$ which divides n then it means that n is prime and we will return True.



```
Input_Data = int( input("Enter a number") )
if Input_Data > 1 :
  for i in range( 2, Input_Data ):
    if Input_Data % i == 0 :
      print( Input_Data, "is not a prime number" )
      break
  else:
    print( Input_Data, "is a prime number" )
else:
  print( Input_Data, "is not a prime number" )
```

Exit

trinket Run

Powered by trinket

Enter a number 7

7 is a prime number

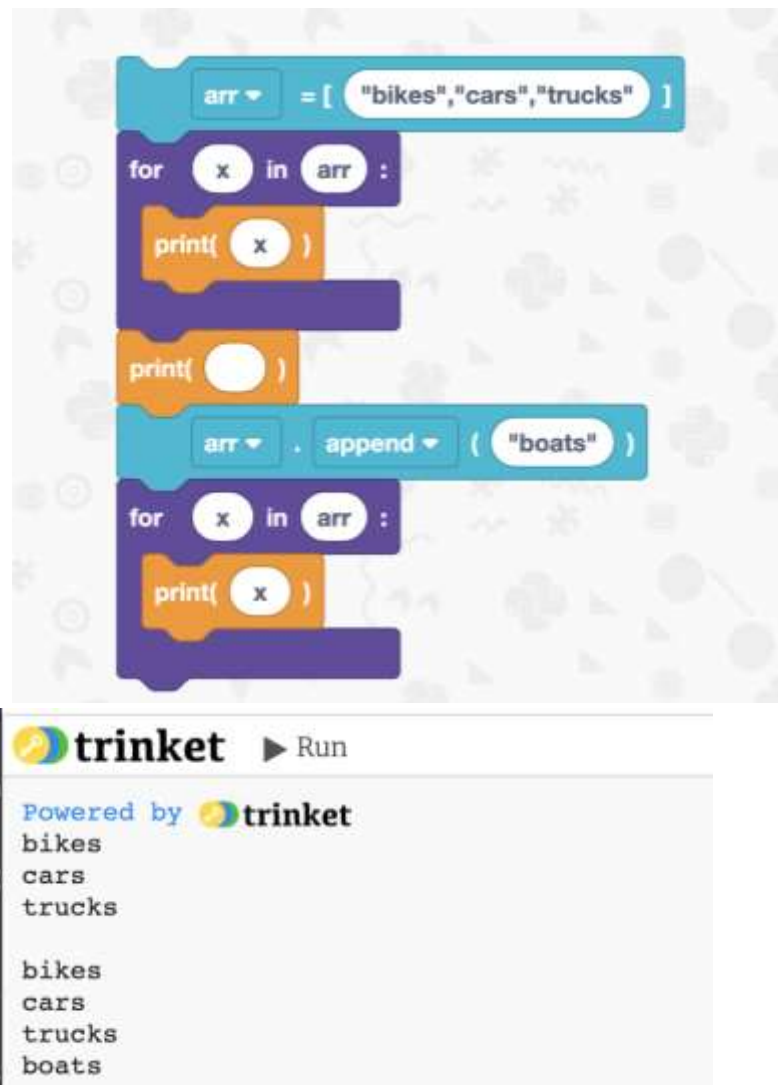
Arrays and Lists

-> List - Declaration and printing

Here we are declaring a list *arr* which contains - bikes, cars, and trucks. We use a for loop to print the contents of the list

Here we are using variable *x* while iterating through the for-loop. *x* corresponds to every element in *arr*.

Then we are adding one more element "boats" to the list, and printing the list again, which will now have 4 elements. Note: When we provide string input to the list, each element must be within quotation marks (Eg: "bikes", ""boats"")



```
arr = [ "bikes", "cars", "trucks" ]

for x in arr :
    print( x )
    print( )

arr.append( "boats" )

for x in arr :
    print( x )
```

trinket Run

Powered by trinket

bikes
cars
trucks

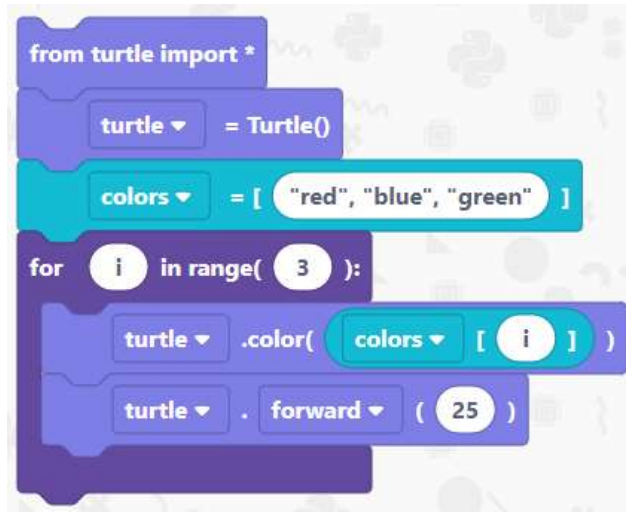
bikes
cars
trucks
boats

***** Try it yourself*****

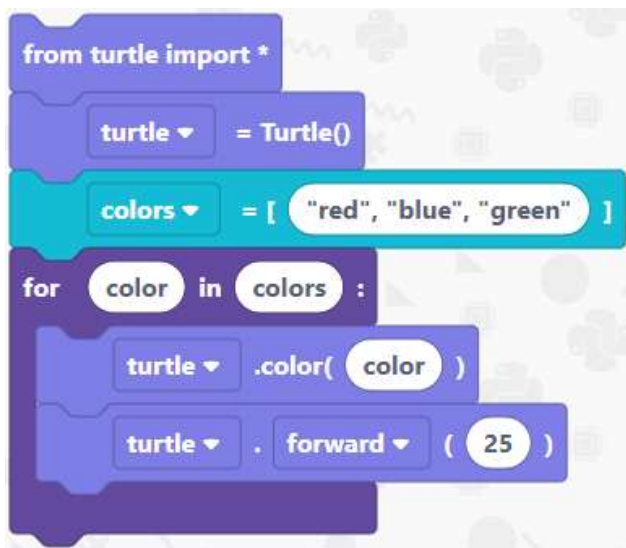
Try using *arr.insert*, *arr.pop*, and *arr.remove* methods provided in the block dropdown and observe the changes.

-> Example of lists with Turtle Library:

Below are two examples of using lists with turtle objects. Both move a turtle 25 pixels forward three times, while setting a new line color each iteration. The only difference is the type of for-loop used.

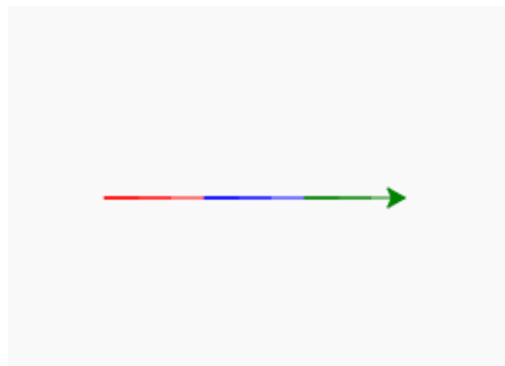


This program uses a for loop which iterates across a specified range (0 to 2, where the outer boundary, 3, is excluded). The color is set by calling `colors[i]`, where *i* is the index of the element.



This program uses a for loop which creates a variable `color` that stores each **element of the list `colors`** for each iteration. So, for iteration 1, it will store `red` in the variable `colors`, `blue` the second iteration, and `green` for the third.

Example output:



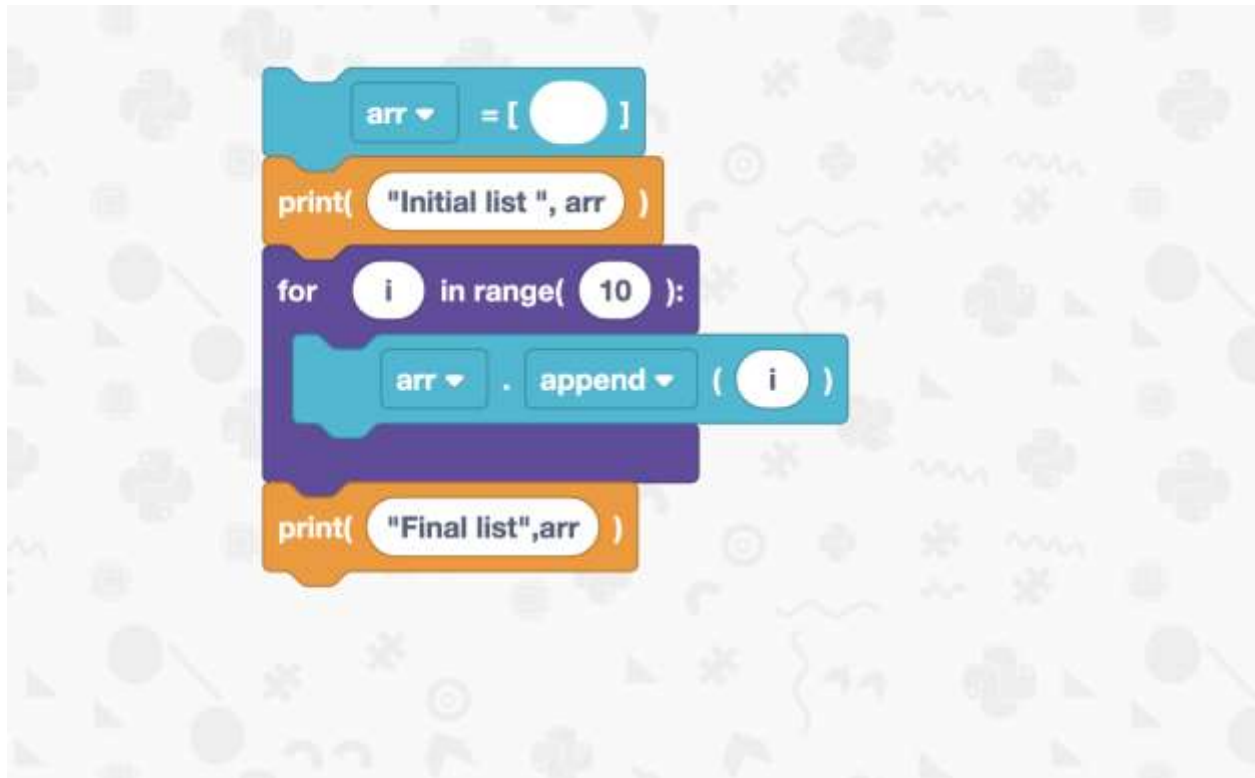
As you can see, the color was changed 3 times using the colors defined in the list.

-> List - Appending to a list

Here we declare a list *arr* as empty and print it.

Using a for loop in range 0 to 10, we append each of these values to the list.

After the for-loop, we print the list again and we can see all the values in the list.



****Try it yourself****

Using the turtle library, create a program which will allow the user to input 5 colors for the turtle's path, then display it by moving the turtle 25 pixels for each color inputted.

To do:

- Create a new empty list called *"colors"*
- Create a for loop which loops 5 times asking the user to enter a color each iteration. Append the user input to the *colors* list.
- After the user has entered 5 colors, iterate through the *colors* list again, and move the turtle as described above using the colors in the list.

Example output:



Powered by  trinket

```
Please enter a color: red
Please enter a color: green
Please enter a color: blue
Please enter a color: purple
Please enter a color: orange
```

Part 2 - Advanced Concepts

Few Important concepts to know before we move ahead:

Definitions/Functions:

```
def my_function():  
    print("Hello from a function")
```

A *def(definition)* is a block of code that can be reused multiple times. To call the block of code above we use

```
my_function()
```

This will print “Hello from a function” on the screen.

Scope of variables:

```
a=10  
def my_function():  
    b=20  
    print(a) # prints 10  
    print(b) # prints 20  
  
print(a) # prints 10  
print(b) # error - because b is not in scope
```

It is important to pay attention to the scoping of the variables. Variables declared inside a definition are local to the definition and cannot be called outside the definition.

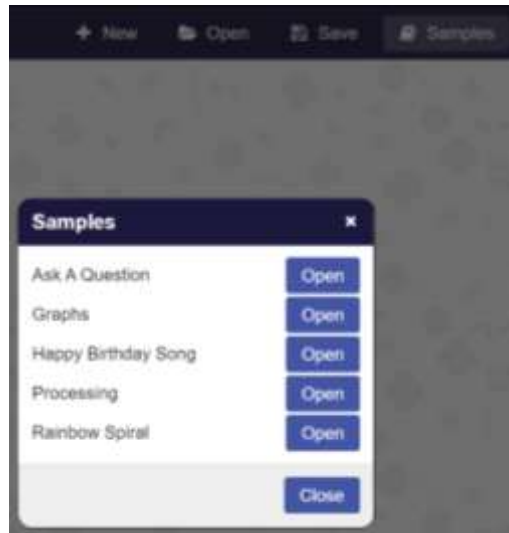
Imports:

We can *import* another python code or module using the *import* keyword in Python. We can find various imports under the import tab of EduBlock. These imported files contain blocks of code that can be used easily and the user does not have to write

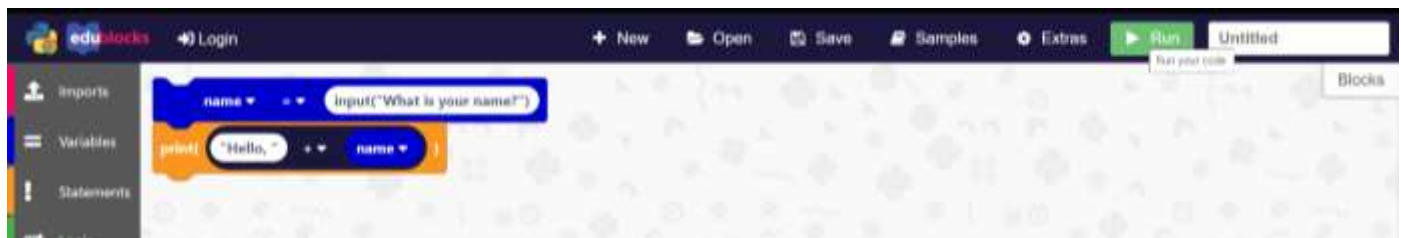
-> How to run Sample Examples:

A. ASK A QUESTION

1. Select Samples -> Select Ask a Question -> Click on the blue Open button



2. This is what it should look like and Select Run to see the output

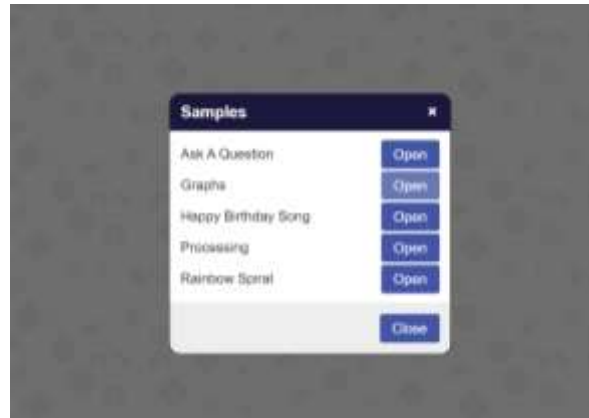


3. The Output window



B. GRAPHS:

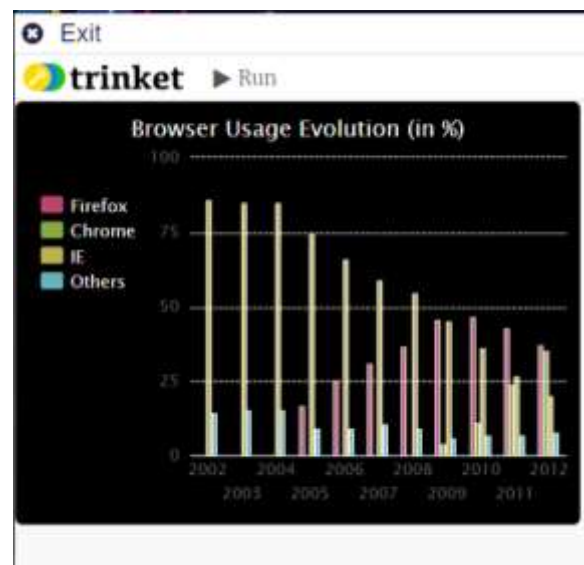
1. Select the Graphs from Samples -> Click Open



2. This is what the sample should look like



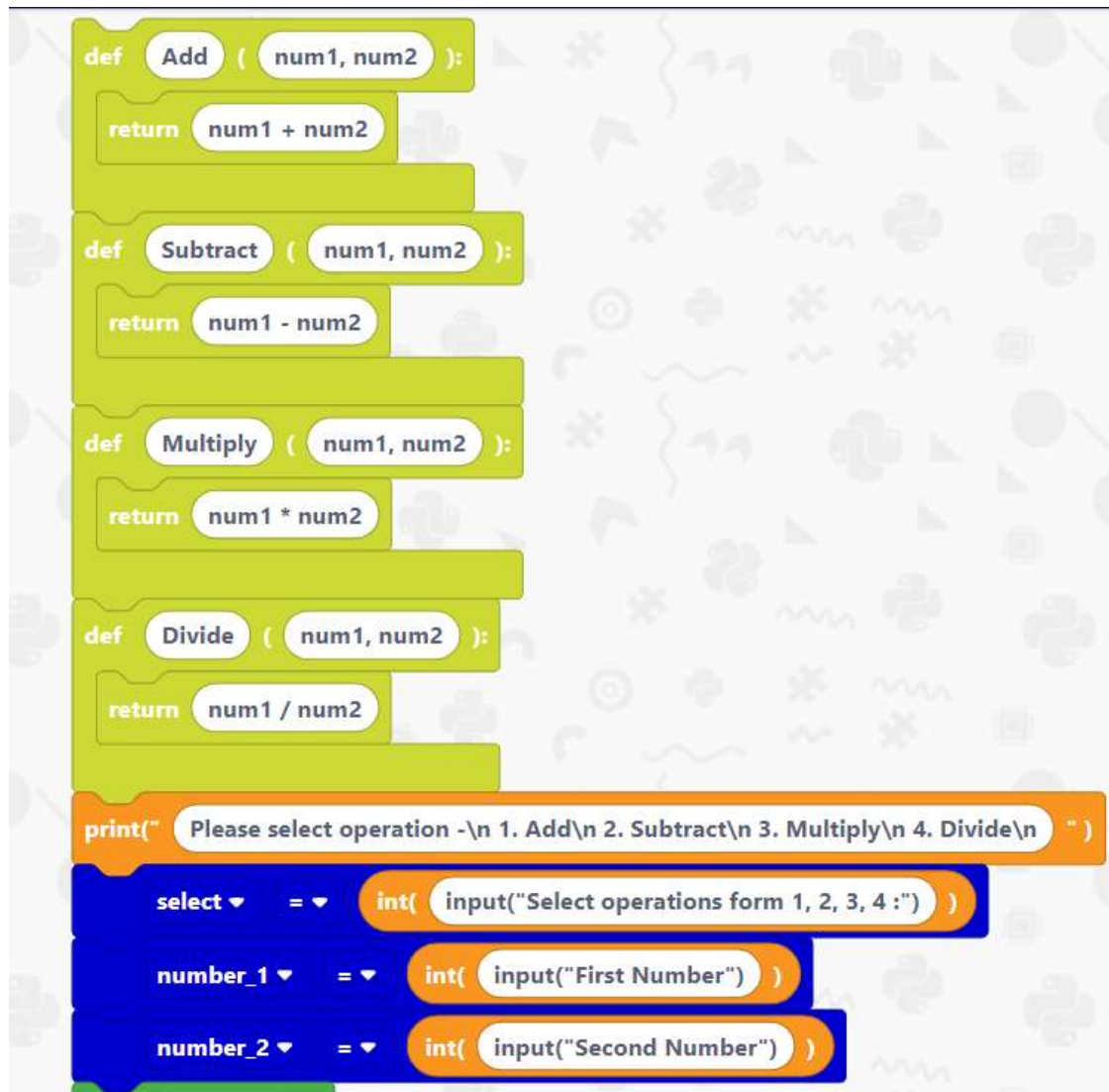
3. The output looks like this

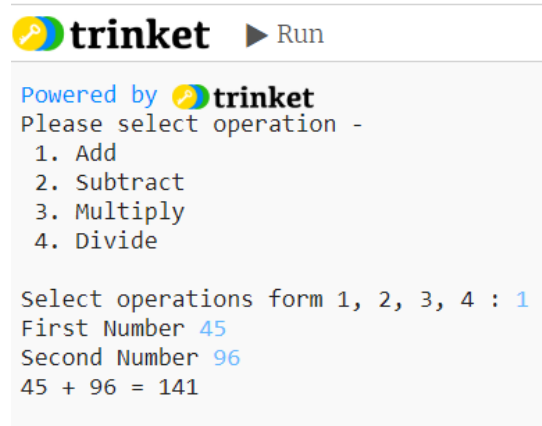


Example 1: Basic Calculator - Add, Subtract, Multiply and Divide

Imagine you are tasked to create a basic calculator capable of doing “Add”, “Subtract”, “Multiply”, and “Divide”.

Think about the steps and try to design a block program to perform the tasks. Here is a sample program that could be a possible solution. Analyze this one and compare it to yours and review the differences.





Example 2: Temperature Converter

How do you calculate temperature conversion?

F° to C°: Fahrenheit to Celsius Conversion Formula

To convert temperatures in degrees Fahrenheit to Celsius, subtract 32 and multiply by . 5556 (or 5/9).

The purpose of the program is to convert temperatures between Celcius, Kelvin, and Fahrenheit

a) The first step is to provide options to the user

Option 1: Celsius to Fahrenheit and Kelvin

Option 2: Fahrenheit to Celsius and Kelvin

Option 3: Kelvin to Celsius and Fahrenheit

We will be letting the user provide the option and accordingly call the corresponding code block
Corresponding code is in the form of if-else block in the option function.

b) Each of the functions that follow the option function is code blocks which convert given temperature to the required temperatures and displays them on the screen

Celsius to Fahrenheit - $((9 * c) / 5) + 32$

Celsius to Kelvin - $c + 273$

Fahrenheit to Celsius - $(5 * (f - 32)) / 9$

Fahrenheit to Kelvin - $c + 273$

Kelvin to Celsius - $k + 273$

Kelvin to Fahrenheit - $((9 * c) / 5) + 32$



Output:

Exit

trinket Run

Powered by trinket

1: Celsius To Fahrenheit and Kelvin
2: Fahrenheit To Celsius and Kelvin
3: Kelvin To Celsius and Fahrenheit

Enter Option : 1

Enter Celsius (Degree) : 32

Temperature : 32.0 Degree Celsius and 89.6 Degree Fahrenheit and 305.0 Kelvin

Example 3: Roots of a quadratic equation $ax^2 + bx + c$

In algebra, a quadratic equation is any equation that can be rearranged in standard form as where x represents an unknown, and a , b , and c represent known numbers, where $a \neq 0$. If $a = 0$, then the equation is linear, not quadratic, as there is no term.

$$ax^2 + bx + c = 0$$

a, b, c = known numbers, where $a \neq 0$

x = the unknown

Example: An equation where the highest exponent of the variable (usually " x ") is a square (2). So it will have something like x^2 . But not x^3 etc. A Quadratic Equation is usually written $ax^2 + bx + c = 0$. Example: $2x^2 + 5x - 3 = 0$.

This program is to find the roots of a quadratic equation, given a , b and c .
The steps followed are:

Step 1: Take a , b and c as inputs from the user

Step 2: Calculate $r = b^2 - 4ac$

Step 3:

- If $r > 0$ then the equation has 2 roots.
They are calculated as $x_1 = \frac{((-b) + \sqrt{r})}{(2*a)}$
 $x_2 = \frac{((-b) - \sqrt{r})}{(2*a)}$
- If $r == 0$ then the equation has 1 root calculated as $(-b) / 2*a$
- If $r < 0$ then the equation has no roots

The conditions in Step 3 are defined in the program in the if-else blocks after the calculation of r .



✕ Exit

trinket ▶ Run

Powered by trinket

```

Quadratic function : (a * x^2) + b*x + c
a: 1
b: -3
c: 1
There are 2 roots: 2.618034 and 0.381966

```

Example 4: Simple Pie chart of World population according to continents

This program creates a simple pie chart of world population distribution according to continents. This program uses an import pygal

Step 1: import pygal from the imports tab

Step 2: From the graphs tab, drag the first definition provided.

Step 3: We have named the chart as **pie_chart** (give any suitable name) and selected the value **Pie** from the drop-down menu.

Step 4: From the Graphs tab, pull the *title* block. Change the variable name and enter a suitable title to be displayed.

Step 5: From the Graphs tab, pull the *add* block. Change the variable name and enter appropriate placeholders for the pie chart (Eg: Asia - 59.69)

Step 6: Repeat Step 5 for multiple values.

Step 7: From the Graphs tab pull the *render* block. Calling the render function is what makes the graph appear on the screen.



Current Population (in %)

