
PROGRAMMING
Python for Kids
The Python for kids course is designed to introduce students to the world of programming using the Pictoblox platform
Learn the basics of coding with Python programming. We are going to use PictoBlox as the tool for programming. Learn concepts of algorithm, loops, sequencing, conditional statements, operators, list, and functions. Learn how to make games with Python in an interactive way.
Start Course ›

Course objectives

By the end of this course, you will be able to:

I. **Learn the basics of programming and Python.**

II. **Find your way around and use Pictoblox easily.**

III. **Create and use variables, data types, and simple rules in Python.**

IV. **Make and use step-by-step instructions to solve problems.**

V. **Combine Python code with Pictoblox blocks for more features.**

VI. **Fix and debug code to make sure it works right.**

VII. **Use programming skills to create real-world projects.**

VIII. **Build fun projects like games, animations, and simple apps.**

Introduction to Pictoblox and Python.

At the end of this lesson, you will:

- Know what programming is and its importance.
- Understand the features and capabilities of the Pictoblox platform.
- Set up the Pictoblox environment for Python programming.
- Navigate the Pictoblox interface and familiarize with its tools and functions

Introduction to Programming

What is programming/coding

Coding is just like solving a math problem. There may be many ways to solve a problem. Similarly, **there could be more than one way to write code for the same task.** Just like solving any other problem, some coding approaches are more efficient than others.

Why is programming important

Programming fosters creativity, reasoning, and problem-solving in you. The programmer gets the opportunity to create something from nothing, use logic to turn programming constructs into a form that a computer can run, and, when things don't work quite as well as expected, use problem solving to figure out what has gone wrong.

How do we programme?

A programming language is simply a particular way to talk to a computer—a way to use instructions that both humans and the computer can understand. We are going to use python language to make programs. Python is an easy-to-learn programming language that has some really useful features for a beginning programmer.

Installing the software

Installing the Software

To begin your programming journey with PictoBlox, you need to first, well, install it. Follow our instructions given below carefully and you'll be well on your way!

Windows Installer (.exe)

STEP 1: Download the Pictoblox Installer (.exe) for Windows 7 and above.

Windows Installer 64-bit

Windows Installer 32-bit

STEP 2: Run the .exe file.

Some of the device gives the warning popup. You don't have to worry, this software is harmless. Click on **More info** and then click on **Run anyway**.

STEP 3: Rest of the installation is straight forward, you can follow the popup and check on the option appropriate for your need.

macOS Installer

STEP 1: Download the Pictoblox Installer (.dmg).

macOS Installer

STEP 2: Run the .dmg file.

LESSON 2

Animations with python

Learning Outcomes

At the end of this lesson, you'll be able to:

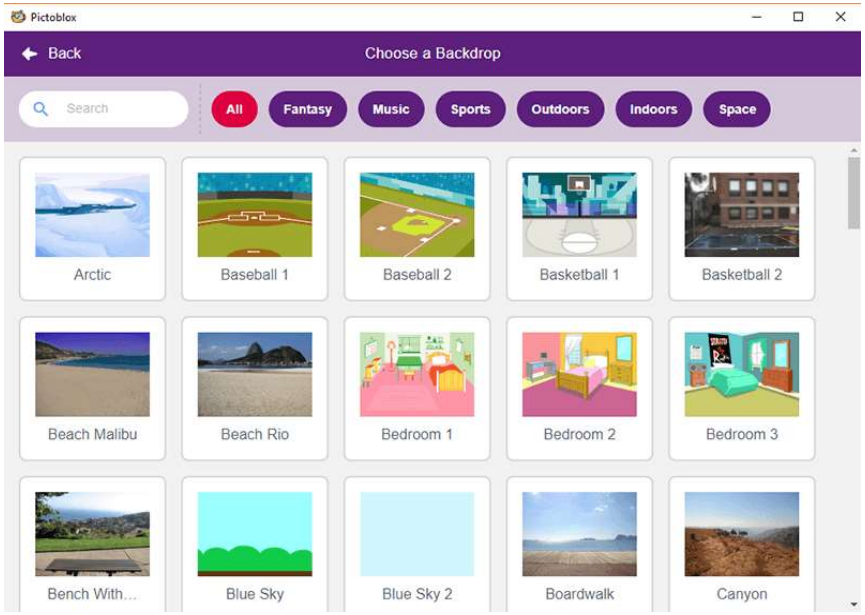
1.Know Python coding environment and how to use it.

2.Know about various functions used to control sprite.

3.Code on your own to make your sprite Tobi walk.

What is a Backdrop?

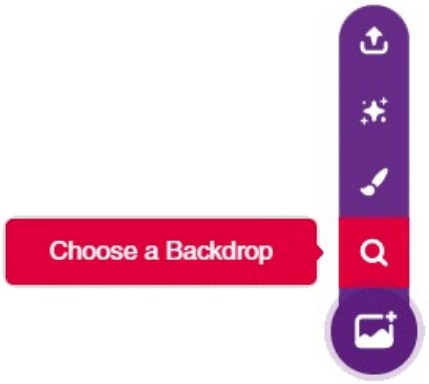
A **backdrop** is one of the many frames, or backgrounds, that a Stage can have. The Stage can change its look to any of its backdrops.



Choosing a Backdrop

You can choose a backdrop from

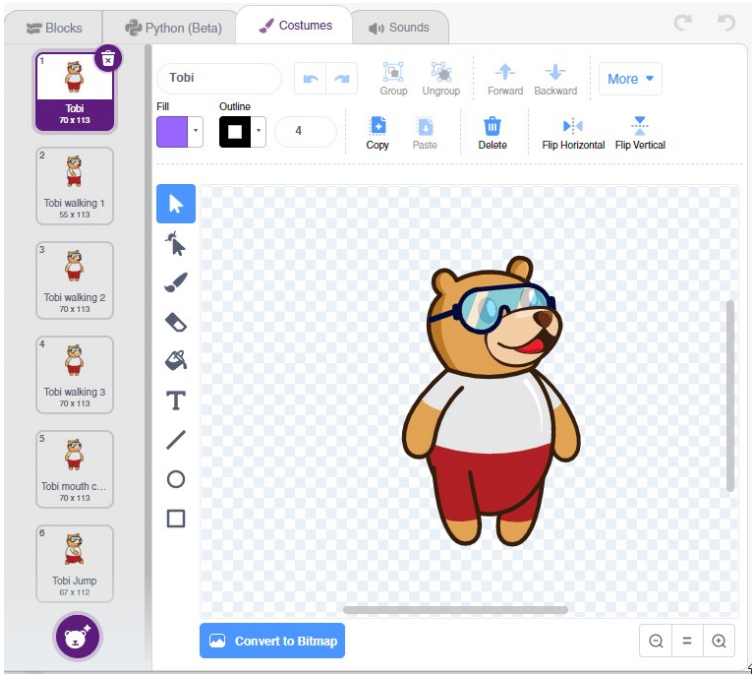
- the backdrop library
- uploading a file from the computer
- creating one using the paint editor



Click Choose the **Backdrop** and select any backdrop you want. We're choosing Blue Sky.

What are Costumes?

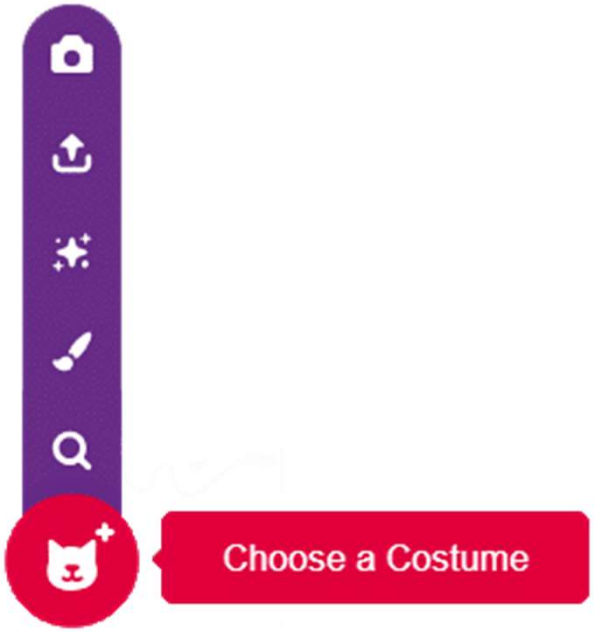
A **costume** is one out of possibly many “frames” or alternate appearances of a sprite. Sprites can change their look to any of its costumes. Every sprite has at least one costume.



One of the most common uses of costumes is to make an animation for a game. You can use a number of costumes to create one single animation. The costumes available for the sprite are shown in the **Costume** tab, next to the **Code** tab on the extreme left.

How to Create Costumes?

There are four ways of getting a costume for a sprite or stage.
1.From the costume library
2.Drawing one yourself using the inbuilt paint editor
3.Getting an image or multiple images from your desktop
4.Taking an image using a webcam



Functions to Control Sprites

A function is a reusable block of code that performs a specific task. Functions are essential for organizing code into modular and manageable pieces. They help improve code readability, promote code reuse, and make it easier to maintain and debug programs.

1.Function Definition: It starts with the keyword `def`, followed by the function name, parentheses, and a colon. The function name should follow Python's variable naming conventions.

2.Parameters: Inside the parentheses, you can declare parameters (inputs) that the function accepts. Parameters are placeholders for values that will be passed when the function is called.

3.Function Body: The code inside the function is indented and defines the tasks the function performs. It can include variable declarations, conditionals, loops, and other statements.

There are a lot of functions with which we can control the different aspects of Sprite. We will look at the few important functions that will be useful to create an animation.

1.Move

1. move()

This function is used to move the sprite a certain number of steps forward. It takes only one input:

1.Number of Steps – Integer

Example

```
sprite = Sprite('Tobi')

sprite.gotoxy(0, 0)
sprite.setrotationstyle('left-right')

while 1:
  sprite.move(3)
  sprite.bounceonedge()
```

2. bounceonedge()

The function checks to see if its sprite is touching the edge of the screen with the move function and if it is, the sprite will point in a direction that mirrors the direction from which it was coming. It uses a line perpendicular to the edge to determine the reflection angle.

Example

```
sprite = Sprite('Tobi')

sprite.gotoxy(0, 0)
sprite.setrotationstyle('left-right')

while 1:
  sprite.move(3)
  sprite.bounceonedge()
```

3. setrotationstyle()

The function changes the rotation style of the sprite in-project. There are three options for this function:

1.all around: All around means the sprite can face any of the 360 degrees. It is the default.

2.left-right: Left-right means sprite can only face left or right, and any other directions are rounded. The sprite will also be horizontally flipped when facing left in the left-right style.

3.don't rotate: Don't rotate means that the sprite always faces as in 90°.

It takes only one input.

4. switchcostume()

This function is used to switch the sprite's costume to a specific costume. It takes only one input:

1.Costume Name – String

Example

```
sprite = Sprite('Tobi')

sprite.switchcostume("Tobi walking 1")
sprite.say("Tobi Walking 1", 2)

sprite.switchcostume("Tobi walking 2")
sprite.say("Tobi Walking 2", 2)
```

6. gotoxy()

This function is used to change the sprite's specified x and y coordinates on the stage. It takes two inputs:

1.X Position – Integer from -240 to 240

2.Y Position – Integer from -180 to 180

Example

```
sprite = Sprite('Tobi')

sprite.gotoxy(0, 0)
sprite.say('This is Center', 2)

sprite.gotoxy(100, 100)
sprite.say('This is Top Right', 2)

sprite.gotoxy(-100, -100)
sprite.say('This is Bottom Left', 2)
```

5. nextcostume()

The function changes its sprite's costume to the next one in the costume pane, but if the current costume is the last in the list, the function will loop to the first.

Example

```
sprite = Sprite('Tobi')

sprite.switchcostume("Tobi walking 1")
sprite.say(sprite.costume("name"), 2)

sprite.nextcostume()
sprite.say(sprite.costume("name"), 2)

sprite.nextcostume()
sprite.say(sprite.costume("name"), 2)

sprite.nextcostume()
sprite.say(sprite.costume("name"),
```

ACTIVITY:Lets Code

The code is pretty simple, let’s get straight into it, follow the below steps:

First, select the Tobi.py file from the Project Files section and by default, the syntax will be written in sprite as an object.

sprite = Sprite('Tobi’)

We need to change the sprite’s position along the x-axis and y-axis, for that we need to include gotoxy() function in a below-given manner:

sprite.gotoxy(0, -100)

Next, we will set up the rotation style to “left-right”.

sprite.setrotationstyle("left-right")

Algorithms & Flowcharts

Learning Outcomes

At the end of this lesson, you'll:

1. Know what is an algorithm and how to write it.
2. Know about what a flowchart is and what the symbols in the flowchart mean.

An algorithm is like a set of instructions you follow to complete a task. For example, your morning routine for school can be seen as an algorithm. First, you wake up when the alarm rings. Then, you go to the bathroom to brush your teeth. Next, you get dressed by choosing and putting on your clothes. After that, you eat breakfast in the kitchen. Then, you pack your school bag with all the necessary items like books and your lunchbox. Finally, you put on your shoes and leave for school. This step-by-step process ensures you are ready for school, just like a computer follows a program's instructions.

An **algorithm** is defined as the step-by-step plan to solve the problem for a given problem statement.

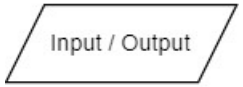
What is a Flowchart?

A flowchart is a diagrammatic representation of the step-by-step plan to be followed for solving a task/problem statement.

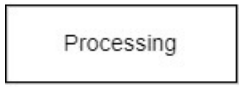
Symbols used in a flowchart

1.Terminal: Indicates start / stop / halt

2.Input / Output: Indicates instructions that either take inputs or display output.



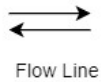
3.Processing: Indicates instructions that represent computation.



4.Decision: Indicates decision-based operations such as Yes/No, or true/false.

5.Connectors: Complex flowcharts which span over more than a page are connected via a connector.

6.Flow lines: Indicates the direction of flow of sequence in a flowchart.



Variables and Arithmetic Operators

By the end of this lesson ,you will be able to:

1. Know about variables and why you need them in Python programming.

2. Know about how you can create and manipulate variables in different ways.

3. Know about the arithmetic operators in Python programming.

4. Code simple programs in Pictoblox to experiment and test your understanding of variables and arithmetic operators.

Why Do You Need Variables?

Let's say you have a special notebook where you keep track of your favorite games. One day, you write down "Minecraft" as your favorite game. A few months later, you discover a new game called "Among Us," and now it becomes your favorite. You erase "Minecraft" and write "Among Us" in its place. In programming, a variable is like that entry in your notebook. It's a name that can hold different values at different times. The name of the variable stays the same, but the information it holds can change whenever you need it to.

What are Variables?

The notebook that is used to keep track of your favorite games can keep track of several other games. In computer programming terms, the notebook can be compared to *variables*.

A ***variable*** is something that can take or store different values as the program is executed.

Data Types in Python

Variables are the values that are acted upon. Every value needs to be assigned to a specific data type to make the variable more readable by a computer.

Data type identifies the type of data that the declared variable can hold. Thus, it indirectly helps the computer to understand what operations need to be performed on those variables.

Let us now understand what are the **common data types** that we can use in programming:

- Integer
- Floating-point number
- String
- Boolean

Integer Data Type

Integer data type variables store integer values only. They store whole numbers which have zero, positive and negative values but not decimal values.

Multiple programming languages support different syntax to declare an Integer variable.

If a user tries to create an integer variable and assign it a non-integer value, the program returns an error.

```
#Example of declaring an Integer variable:

a = 2
b = -156

print(a)
print(b)
```

Variables of the integer data type are only capable of holding single values. These variables are not capable of holding a long list of values.

Floating Point Number Data Type

Floating-point numbers are used to store decimal values. They hold real numbers with decimal values.

Depending on the programming language, the syntax to declare floating-point variables changes.
We can convert float to an integer using the **int()** function.

```
#Example of declaring an Float variable:

a = 2.5
b = -1.5

print(a)
```

String Data Type

To extend the character data type, a user may have a requirement to store and perform an operation on a sequence of characters. In such cases, the String data type is present to fit the gap. The String data type stores value in a sequence of characters i.e. in String format.

```
#Example of declaring an String variable:

a = 'I am '
b = 'Tobi'
c = 5

print(a)
print(a, b)
print("Number c is ", c)
```

Boolean Data Type

There is a subtype of Integer Data Type called “Boolean Data Type”, which stores values in Boolean type only i.e. “true” or “false”. Users can choose between the data types for variables as per program needs and assign variables an appropriate data type.

Boolean is a subtype of integer data type. It stores true and false where true means non-zero and false means zero.

```
#Example of declaring an Boolean
variable:

a = True
b = False

print(a)
print(b)
```

Naming Rules

As we have understood till now, **variables are basically like nouns in a programming language**. **Every variable in a program is unique.** To identify these variables uniquely, the user needs to allocate them a unique name. This name acts as an identifier for that variable. In programming, a user is not allowed to use the same name of a variable more than once.

Naming variables make it easier to call them while performing operations. The name of a variable also suggests what information the variable contains.

Below are some rules for naming a variable:

•A variable name cannot start with a number, it must start with an alphabet or the underscore (_) sign

•A variable name is case-sensitive. **Sum** and **sum** are different variables

•A variable can only contain alphanumeric characters and underscore

Arithmetic Operators

Operators

Operators are special symbols that represent computation. They are applied to operand(s), which can be values or variables. The same operator can behave differently on different data types. Value and variables when used with an operator are known as operands.

Operators are categorized as
1.Arithmetic
2.Relational
3.Logical
4.Assignment.

Mathematical/Arithmetic Operators

#	Symbol	Description	Example 1	Example 2
1	+	Addition	print(60 + 40) >>100	print("Good" + "Morning") >>GoodMorning
2	-	Subtraction	print(60 - 40) >>20	print(30 - 80) >>-50
3	*	Multiplication	print(60 * 40) >>2400	print("Good" * 3) >>GoodGoodGood
4	/	Division	print(17 / 5) >>3.4	print(3.4 / 1.7) >>2.0
5	//	Integer Division	print(7.0 // 2) >>3.0	print(3 // 2) >>1
6	%	Remainder / Modulo	print(17 % 5) >>2	print(23 % 2) >>1
7	**	Exponentiation	print(2 ** 3) >>8	print(16 ** 0.5) >>4.0

Activity 1: Addition Bot

Taking Inputs with Sprite

In a program, it's very important to take the inputs from the user. For that there are 2 important functions we will use:

1.input(): This function makes the sprite ask the question to the user.

2.answer(): This function stores the value entered by the user.

```
sprite = Sprite('Tobi')

sprite.input("Enter Number 1")
a = int(sprite.answer())

sprite.say("You have entered: " + str(a))
```

Let us now create our addition bot, by using the concept of variables and arithmetic operators learned till now. The bot would be responsible for performing an addition operation on two numbers, which the user needs to enter.

Example:

```
sprite = Sprite('Tobi')

sprite.input("Enter Number 1")
a = int(sprite.answer())

sprite.input("Enter Number 2")
b = int(sprite.answer())

sum = a + b

sprite.say("Addition is", 2)
sprite.say(sum)
```

Area Calculator

Let us now make an area calculator bot! The user needs to enter the length and breadth and the bot displays the result as a multiplication of length and breadth.

```
sprite = Sprite('Tobi')

sprite.input("Enter the length")
length = int(sprite.answer())

sprite.input("Enter the breadth")
breadth = int(sprite.answer())

area = length * breadth

sprite.say("Area is" + str(area))
```

Functions in Python

Learning outcomes

1. The usefulness of using functions in code.
2. How to define and call a function.
3. Parameters in a function

Mathematical Functions in Python

A function is a block of code made up of a set of steps that results in a single specific action. The programmer will give this action a simple name. Giving a simple name to a function increases the chances that the set of steps can easily be talked about and reused again and again in the program.

Functions can be categorized as belonging to
1.Modules
2.Built-in
3.User-Defined

Module

A module is a file containing Python definitions (i.e. functions) and statements. The standard library of Python is extended as a module(s) to a programmer. Definitions from the module can be used within the code of a program. A programmer needs to import the module to use these modules in the program. Once you import a module, you can reference (use), any of its functions or variables in your code.

Import is the simplest and most common way to use modules in our code. Its syntax is:

import modulename1 [,modulename2, ———]

Example

```
Example
import math
```

On execution of this statement, Python will
1.Search for the file "**math.py**".
2.Create space where modules definition & variable will be created,
3.then execute the statements in the module.

Now the definitions of the module will become part of the code in which the module was imported. To use/ access/invoke a function, you will specify the module name and name of the function- separated by a dot (.). This format is also known as *dot notation*.

```
value= math.sqrt (25) # dot notation
```

The example uses the sqrt( ) function of module **math** to calculate the square root of the value provided in parenthesis and returns the result which is inserted in the ***value***. The expression (variable) written in parenthesis is known as argument (actual argument). It is common to say that the function takes arguments and returns the result.

Built in Functions

Built-in functions are the function(s) that are built into Python and can be accessed by a programmer. These are always available and for using them, we don‟t have to import any module (file). Python has a small set of built-in functions as most of the functions have been partitioned into modules. This was done to keep core language precise.

Name	Description	Example
abs(x)	It returns the distance between x and zero, where x is a numeric expression.	abs(-45) >>45 abs(119) >>119
max(x, y, z, ...)	It returns the largest of its arguments: where x, y, and z are numeric variables/expressions.	max(80, 100, 1000) >>1000 max(-80, -20, -10) >>-10
min(x, y, z, ...)	It returns the smallest of its arguments; where x, y, and z are numeric variables/expressions.	min(80, 100, 1000) >>80 min(-80, -20, -10) >>-80
cmp(x, y)	It returns the sign of the difference of two numbers: -1 if x < y, 0 if x == y, or 1 if x > y, where x and y are numeric variable/expression.	cmp(80, 100) >>-1 cmp(180, 100) >>1
round(x [, n])	It returns float x rounded to n digits from the decimal point, where x and n are numeric expressions. If n is not provided then x is rounded to 0 decimal digits.	round(80.23456, 2) >>80.23 round(-100.000056, 3) >>-100.0 round (80.23456) >>80.0

Composition

Composition is the art of combining simple function(s) to build more complicated ones, i.e., the result of one function is used as the input to another.

Example

Suppose we have two functions fn1 & fn2, such that

a=fn2(x)

b= fn1 (a)

then call to the two functions can be combined as

b= fn1 (fn2 (x))

User Defined Functions

So far we have only seen the functions which come with Python either in some file (module) or in the interpreter itself (built-in), but it is also possible for a programmer to write their own function(s). These functions can then be combined to form a module which can then be used in other programs by importing them.

To define a function keyword **def** is used. After the keyword comes an identifier i.e. name of the function, followed by a parenthesized list of parameters and the colon which ends up the line. Next follows the block of the statement(s) that are part of the function.

Block of statements

A **block** is one or more lines of code, grouped together so that they are treated as one big sequence of statements while executing.

In Python, statements in a block are written with indentation. Usually, a block begins when a line is indented (by four spaces) and all the statements of the block should be at the same indent level. A block within block begins when its first statement is indented by four spaces, i.e., eight spaces. To end a block, write the next statement with the same indentation before the block started.

Syntax of function

def NAME ([PARAMETER1, PARAMETER2, ...]): #Square brackets include

statement(s) #optional part of statement

Let‟s write a function to greet the world:

```
def sayHello (): # Line No. 1
print “Hello World!” # Line No. 2
```

Parameters and Arguments

Parameters are the value(s) provided in the parenthesis when we write the function header. These are the values required by the function to work.

Let's understand this with the help of a function written for calculating the surface area of a cube. **length** is a parameter to function area.

```
def surfaceAreaCube(length): #
Calculates the surface area of the
cube surfaceArea = 6*length*length
return surfaceArea
```

If there is more than one value required by the function to work on, then, all of them will be listed in the parameter list separated by a comma.

Arguments are the value(s) provided in function call/invoke statement. A list of arguments should be supplied in the same way as parameters are listed. Bounding of parameters to arguments is done 1:1, and so there should be the same number and type of arguments as mentioned in the parameter list.

Lets code

```
sprite = Sprite('Tobi')

def surfaceAreaCube(length):
  # Calculates the surface area of the cube
  surfaceArea = 6*length*length
  return surfaceArea

def volumeCube(length):
  # Calculates the volume of the cube
  volume = length*length*length
  return volume

sprite.input("Enter the side length")
l = int(sprite.answer())

sprite.say("Surface Area is " + str(surfaceAreaCube(l)), 2)
sprite.say("Volume is " + str(volumeCube(l)), 2)
```

Simple Interest Formula

Simple interest is calculated with the following formula:

S.I. = P × R × T, where P = Principal, R = Rate of Interest in % per annum, and T = Time, usually calculated as the number of years. The rate of interest is in percentage r% and is to be written as r/100.

Code

```
sprite = Sprite('Tobi')

def interestCalculator(principal, rate, time):
  amount = principal + principal*rate*time/100

  return amount

sprite.input("Enter the principal amount")
principal = float(sprite.answer())
sprite.input("Enter interest rate")
rate = float(sprite.answer())
sprite.input("Enter the time")
time = float(sprite.answer())

sprite.say("Amount is: "+ str(interestCalculator(principal,rate,time)), 2)
```