

Assignment 1: Introduction to Algorithms

The term "algorithm" refers to a finite set of precise rules or instructions designed to be followed in calculations or other problem-solving operations. Algorithms play a crucial role in various fields, including computer science, mathematics, and artificial intelligence (AI), serving as the backbone for processing data and making decisions. In the realm of AI, one of the ultimate goals is to design a single algorithm capable of solving any task (or at least any task that humans can solve).

This assignment serves as an introduction to the basic concepts of algorithms. You will learn how to develop an algorithmic way of thinking, which involves breaking down complex problems into manageable and reusable steps, and systematically solving them.

Part 1: Algorithmic Thinking in Daily Lives

Algorithmic thinking is integral to our daily lives. For example, the Bass method is a recommended algorithm for brushing teeth, particularly effective in maintaining good oral hygiene and preventing gum disease. To ensure anyone can precisely follow the Bass method, people have developed the following algorithm, consisting of precisely-specified steps:

1. Hold the toothbrush parallel to your teeth with the bristles touching your teeth.
2. Tilt the brush to a 45-degree angle.
3. Angle the brush so the bristles are slightly under the gumline.
4. With firm yet gentle pressure, brush back and forth or use a circular motion 15 to 20 times before moving to the next area.
5. Brush all teeth on the outer surface and then clean the backside of the teeth using the same motions.
6. Hold the toothbrush in a vertical position behind your front teeth and brush up and down, using the bristles on the toe of the brush.
7. Brush the chewing surface of the molars and brush your tongue.

By faithfully executing this algorithm, anyone can effectively perform the Bass method and thoroughly clean their teeth. Similarly, computer programs, including AI programs, are precisely-specified steps that a computer can follow.

iteration is an important part of algorithmic thinking. iteration involves repeating a step multiple times that reduces the effort of describing an algorithm. Identify the iteration in the Bass method algorithm. Indicate the step number as follows:

[Student answer:

Step 4.

]

Now let's begin practicing algorithmic thinking! Choose one of your daily tasks—such as doing laundry, a gym workout, making toast, or any other routine activity—and develop a detailed algorithm for it. Break down the task into clear, sequential steps that can be easily followed.

As you do so, try to consider if someone could follow your steps literally, but not do what you intend (for example, if step 1 above had just said, “Hold the toothbrush parallel to your teeth,” someone may hold the brush in front of their face, but not touch their teeth). Try not to leave any room for interpretation!

[Blank for students to fill out... This will be an open question with no wrong answer.

]

The algorithm might completely fail if alternative interpretation exists when executing the steps. Watch this video of PB&B for what could happen when the descriptions are not precise enough. Identify one step in your algorithm that might have room for interpretation and rewrite the step as follows:

[Student answer:

]

Part 2: Developing Algorithms for Computers

Algorithms should be independent of the language in which they are described. However, for computers to understand and execute these algorithms, they must be written in programming languages.

Let's start by developing an algorithm to control a Turtlebot to draw a square. A Turtlebot can execute only four different actions: moving forward by a specified distance, moving backward by a specified distance, turning left by a specified angle, and turning right by a specified angle.

Putting aside the specifics of any programming language, an algorithm to control the Turtlebot to draw a 50m-by-50m square can be described in English as follows:

1. Move forward 50 meters.
2. Turn left 90 degrees.
3. Move forward 50 meters.
4. Turn left 90 degrees.
5. Move forward 50 meters.
6. Turn left 90 degrees.
7. Move forward 50 meters.

Now we will translate the algorithm into a programming language, called Python. The Turtlebot programming language uses four specific commands to control its actions: `t.forward(x)`,

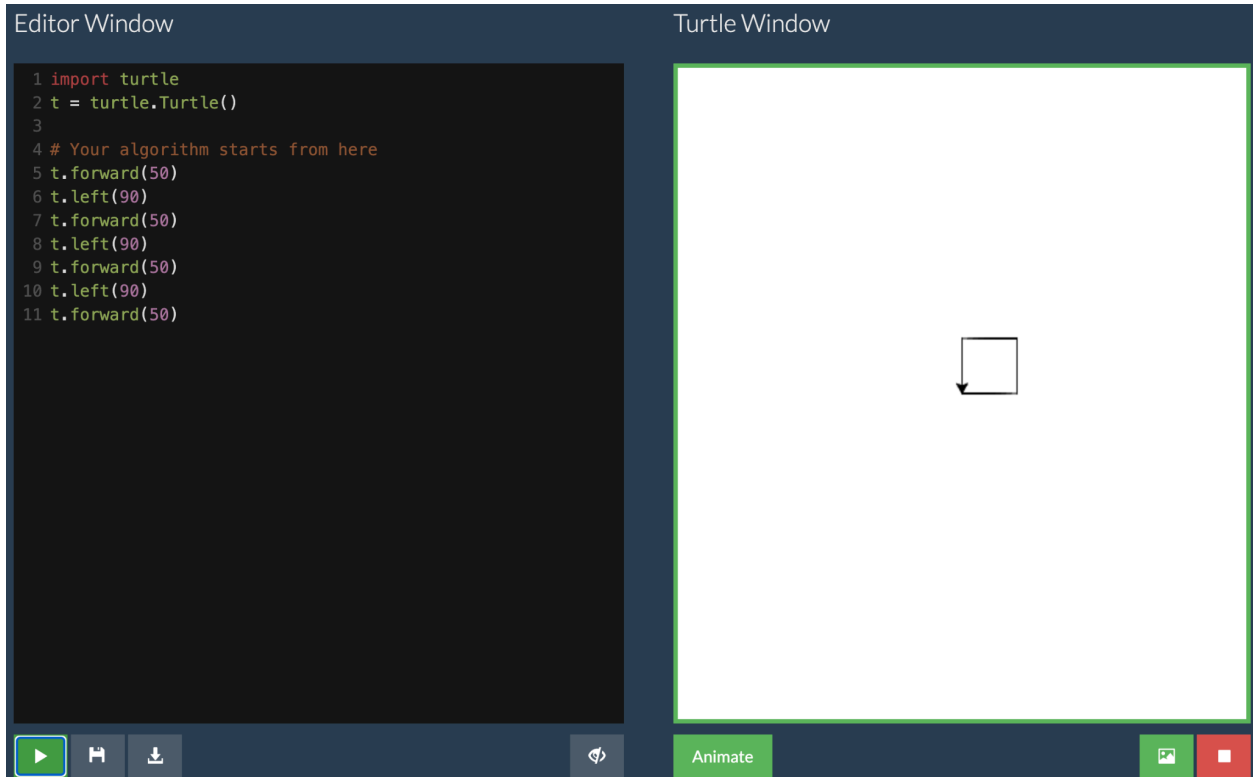
`t.backward(x)`, `t.left(x)`, and `t.right(x)`. These commands correspond to the Turtlebot's four possible actions: moving forward by the specified distance `x`, moving backward by the specified distance `x`, turning left by the specified angle `x`, and turning right by the specified angle `x`. Below is a translation of the algorithm into Python:

```
...  
t.forward(50)  
t.left(90)  
t.forward(50)  
t.left(90)  
t.forward(50)  
t.left(90)  
t.forward(50)  
...
```

Now let's test our algorithm using a simulated Turtlebot in the Python Sandbox. Follow these steps:

1. Open [\[this link\]](#) and copy the algorithm into the Editor Window on the left (ensure you keep the first two lines that set up the Turtlebot).
2. Once the algorithm is entered, click the green play button at the bottom left.
3. Observe the Turtlebot, represented by an arrow, drawing a square pattern in the Turtlebot Window on the right panel.

A successful execution looks like this:



You have already learned how to develop an algorithm, translate the algorithm into a programming language, and test the algorithm. The next step is to repeat those steps for a different algorithm.

Your turn

Develop an algorithm to instruct a Turtlebot to draw an equilateral triangle with each side measuring 50 meters.

(Blank 1: asking students to develop an algorithm in English

1. Move forward 50 meters.
2. Turn left 120 degrees.
3. Move forward 50 meters.
4. Turn left 120 degrees.
5. Move forward 50 meters.

)

(Blank 2: asking students to translate the algorithm into Python

...

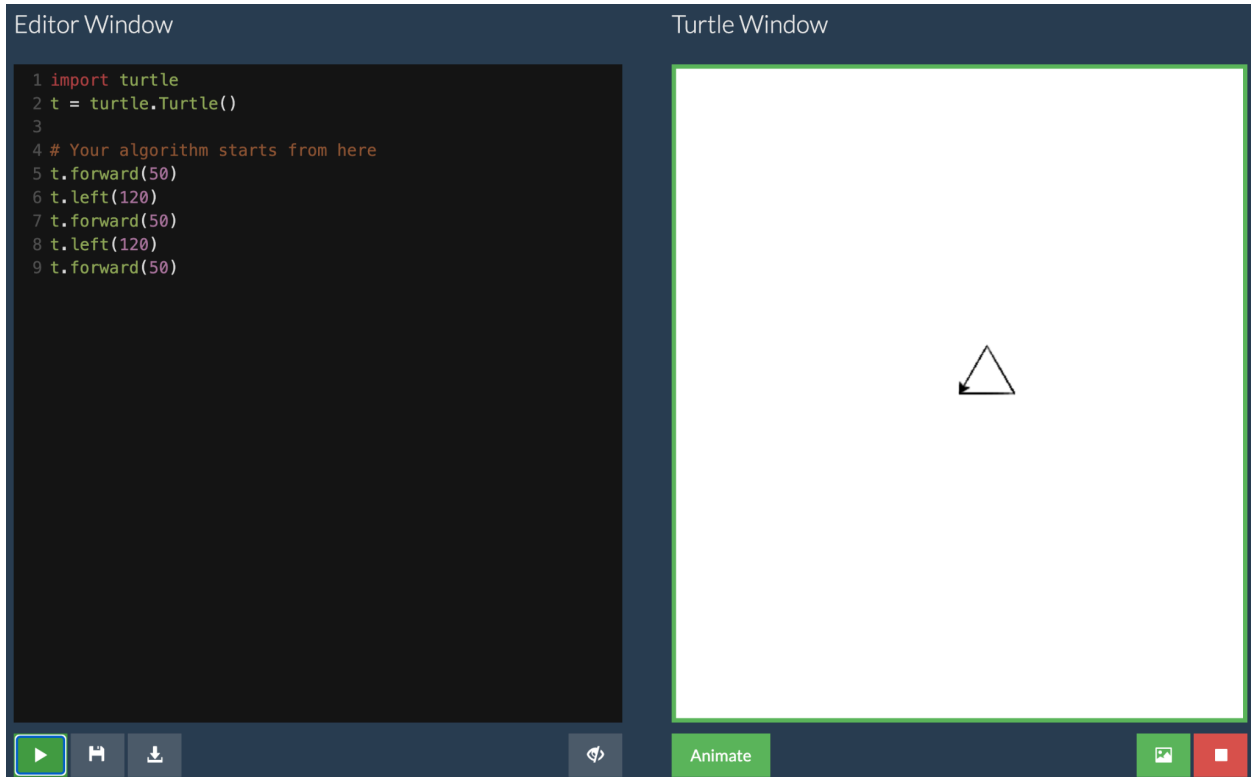
```
t.forward(50)
t.left(120)
t.forward(50)
t.left(120)
```

```
t.forward(50)
```

```
...
```

```
)
```

(Blank 3: upload a screenshot of the test using Python Sandbox



```
)
```

Part 3: Simplifying Algorithms with Iteration

An important aspect of algorithmic thinking is iteration, which involves repeating certain steps that can significantly reduce the effort of designing an algorithm. For example, an algorithm that controls the Turtlebot to draw a square involves repeatedly moving forward and turning left four times. A simplified version of the square-drawing algorithm using iteration can be described as follows:

1. **Repeat the following two steps four times;**
2. Move forward 50 meters;
3. Turn left 90 degrees.

The translation of the above algorithm into Python requires the use of a "for" loop. A "for" loop allows us to repeat a block of code a specific number of times, which is perfect for our iterative

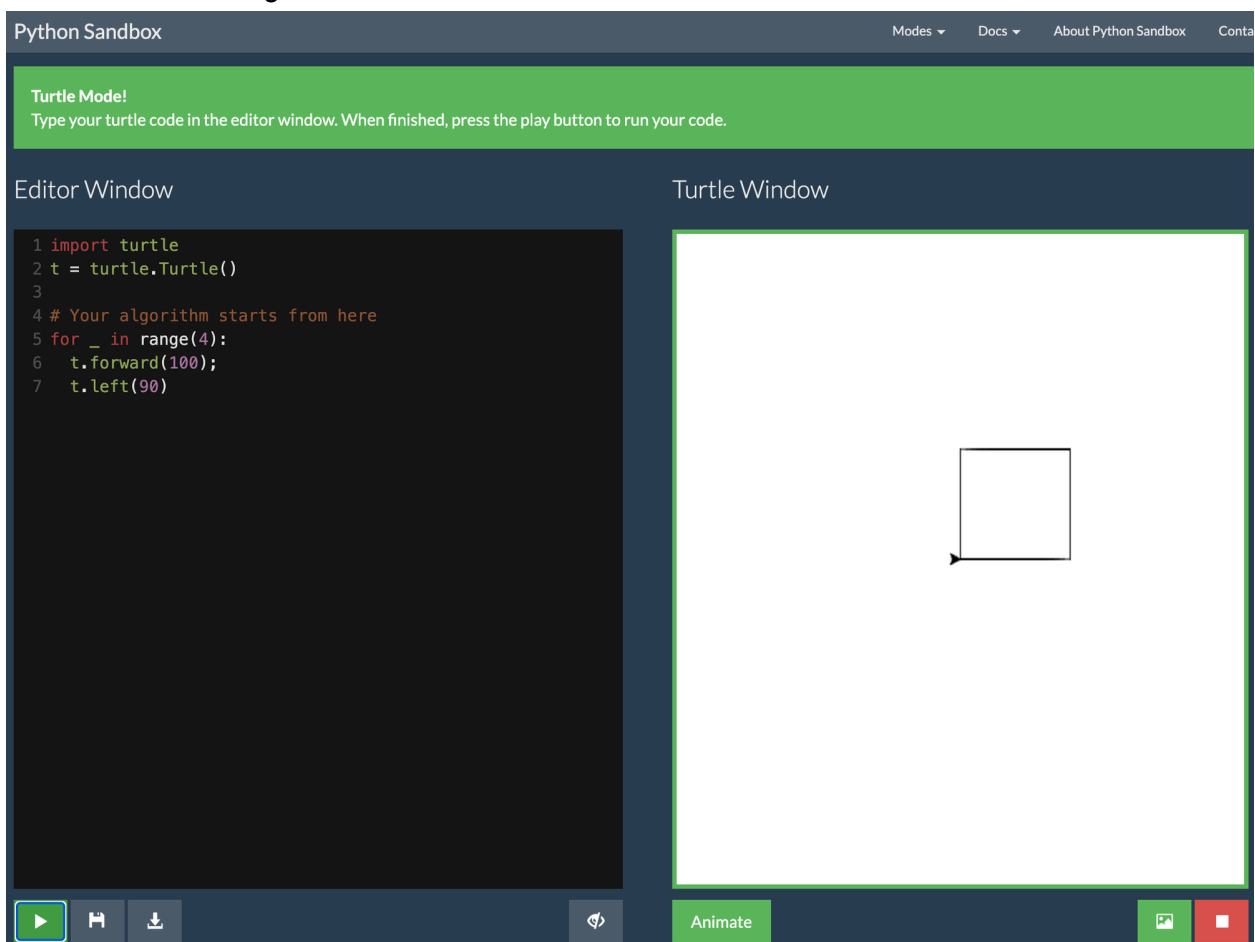
square-drawing algorithm. In Python, the basic “for” loop command `for i in range(n)` repeats following indented commands `n` times. Here is a simple translation of the above algorithm using a "for" loop:

```
for i in range(4):  
    t.forward(50)  
    t.left(90)
```

This algorithm repeats the intended commands `t.forward(50)` and `t.left(90)` four times.

)

Now, let's test our algorithm with iteration:



The screenshot shows a Python Sandbox interface. At the top, there's a green banner that says "Turtle Mode! Type your turtle code in the editor window. When finished, press the play button to run your code." Below this, there are two main windows: "Editor Window" and "Turtle Window". The Editor Window contains the following Python code:

```
1 import turtle  
2 t = turtle.Turtle()  
3  
4 # Your algorithm starts from here  
5 for _ in range(4):  
6     t.forward(100);  
7     t.left(90)
```

The Turtle Window shows a white canvas with a green border. A small black square is drawn in the center of the canvas, with a small black arrow pointing to the right at the bottom-left corner, indicating the turtle's current position and direction.

At the bottom of the interface, there are several control buttons: a play button (green), a home button (blue), a download button (blue), a refresh button (blue), an "Animate" button (green), a screenshot button (green), and a close button (red).

It did a perfect job drawing a square!

Now, can you simplify your algorithm of drawing an equilateral triangle using iteration?
(Blank for students to fill out the python code
...)

```
for _ in range(3):  
    t.forward(50)
```

```
t.left(90)
```

```
...
```

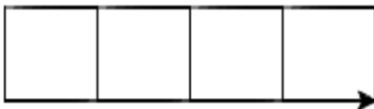
```
)
```

(Blank: upload a screenshot of the test using Python Sandbox)

Part 4: Debugging Your Algorithms

For complex algorithms, it is unlikely that we can create a correct version on the first attempt. This is where debugging comes in. Debugging involves multiple rounds of modifying, testing, and analyzing the algorithm. The value of testing lies not only in verifying the algorithm's correctness but also in identifying issues when the algorithm does not work as expected.

Let's go through a debugging cycle. Consider an algorithm for drawing four squares side-by-side:



We have learned about iteration. To draw four squares side-by-side, someone proposed repeating the square-drawing algorithm introduced in Part 1 four times, with the Turtlebot moving to the starting position of the next square at the end of each iteration. They developed the following algorithm:

- 1. Repeat the following steps four times:**
 - a. Move forward 50 meters.
 - b. Turn left 90 degrees.
 - c. Move forward 50 meters.
 - d. Turn left 90 degrees.
 - e. Move forward 50 meters.
 - f. Turn left 90 degrees.
 - g. Move forward 50 meters.
 - h. Move forward 50 meters to bring the Turtlebot to the starting position of the next square.**

Here is the translation of this algorithm into Python:

```
...
```

```
for _ in range(4):
```

```
    t.forward(50)
```

```
t.left(90)
```

```
t.forward(50)
```

```
t.left(90)
```

```
t.forward(50)
```

```
t.left(90)
```

```
t.forward(50)
```

...

Please test this algorithm using Python Sandbox and describe the issue.

(Blank: student upload test of this algorithm and describe what the issue is)

Modify this algorithm so it can draw four squares side-by-side as expected. Copy your Python code in the text box below and upload a screenshot of the successful test showing the four side-by-side squares.

(Blank 1: student's algorithm in Python code)

(Blank 2: a successful test showing the four side-by-side squares.)