Activity 1.1.8

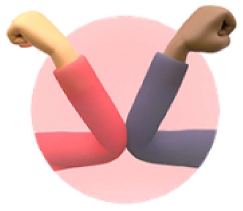
Turtles in Traffic

Distance Learning Support

Check with your teacher about:

☐ What work you need to turn in and how to submit it
☐ Collaboration strategies
☐ If you are using a Chromebook, open a **blank code editor**.

GOALS



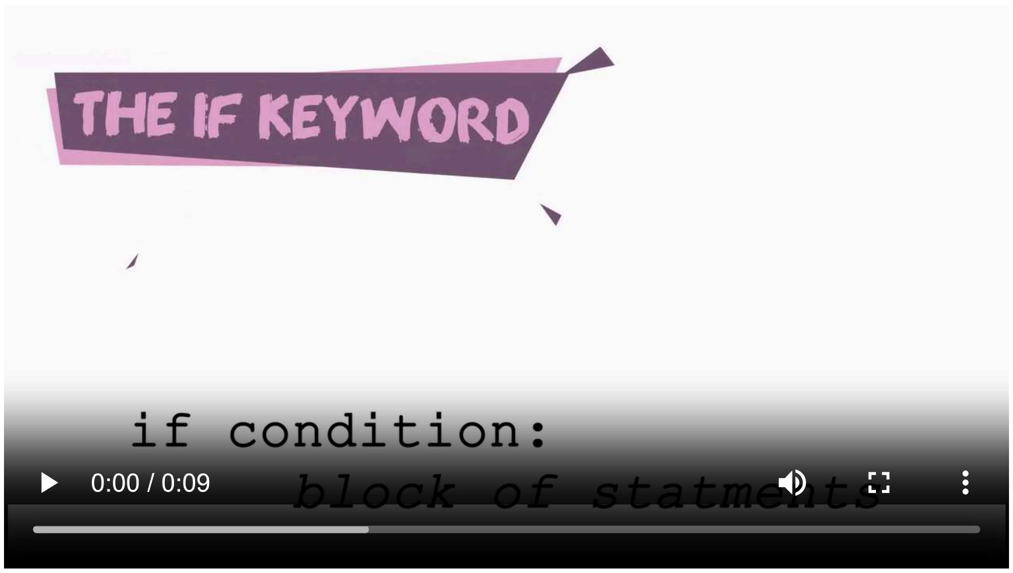
- Use `if` statements to create conditional execution.
- Create an algorithm with nested conditionals and nested iteration to stop turtle movement when they collide.

Conditionals

There are a lot of situations where you only want to execute different statements “if” certain conditions are true. This type of situation is called a **conditional** . Conditionals occur in everyday life too, as you make decisions. For example, think of how you got ready for this morning. Maybe you checked what the weather was outside. “If” it was cold or rainy outside, you grabbed a jacket or heavy shirt. “If” you had plans after school, you grabbed something extra for your backpack. “If” the school’s menu wasn’t to your liking today, you packed your own lunch. “If” your favorite food wasn’t in the fridge, you grabbed something else. A lot of decisions could be made in a short time between waking up and leaving the house.

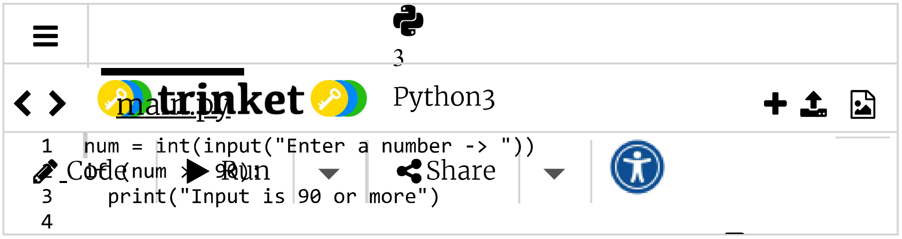
The `if` statement

In *Python*®, the **conditional statement** is called `if`. The *`block of statements`* indented under the `if` statement will only execute if the *`condition`* is true. Otherwise, if it isn't true, all of the code in the block is skipped and not executed. In this way, the conditional runs parts of the program through **selection** . Observe the general *Python* syntax for a simple `if` statement.



`if` Statement Syntax video (The video has no sound.)

1. Examine the following code and make a prediction of the results.

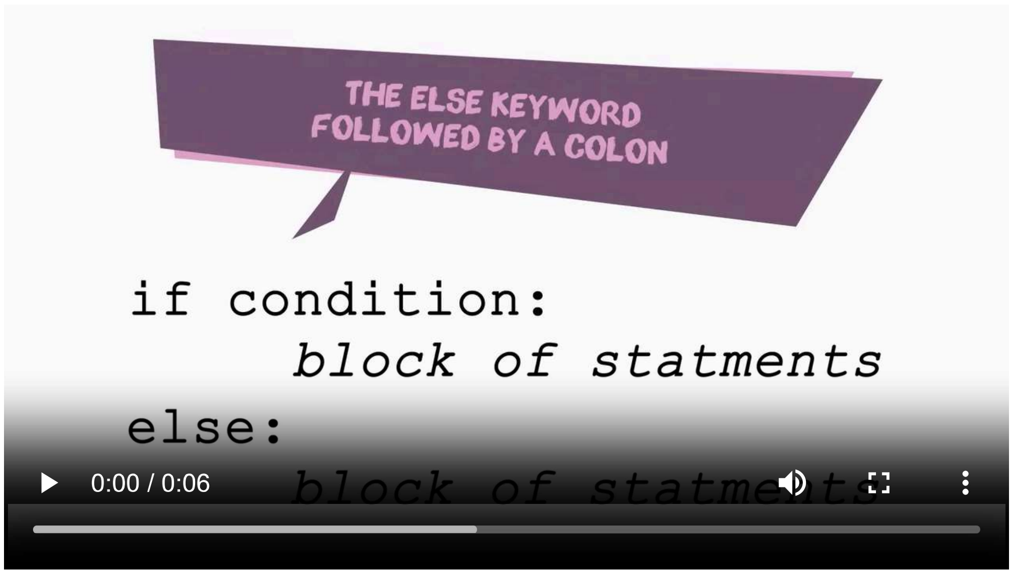


```
num = int(input("Enter a number -> "))
if (num > 90):
  print("Input is 90 or more")

```

2. Run the program and experiment by entering different input numbers and see how the program responds.

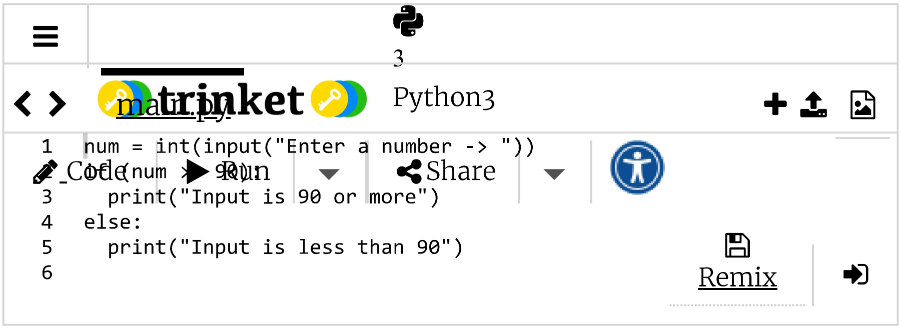
The code will not do anything unless the input is greater than 90. This behavior can be confusing to a user, making them think nothing is happening. Luckily, the `if` statement can expand to include what is called an `else` clause. The `else` clause handles the situation when the condition is not true. If the condition is true, the program will execute the block of statements directly below the `if` statement;

otherwise, or "else," it will execute the block of statements below the `else` clause. The `else` clause serves as a catch-all when no other condition is met.



`else` Clause Syntax video (The video has no sound.)

3 Examine the following code. Predict what it will do based on the inputs.



```
num = int(input("Enter a number -> "))
if(num >= 90):
  print("Input is 90 or more")
else:
  print("Input is less than 90")
```

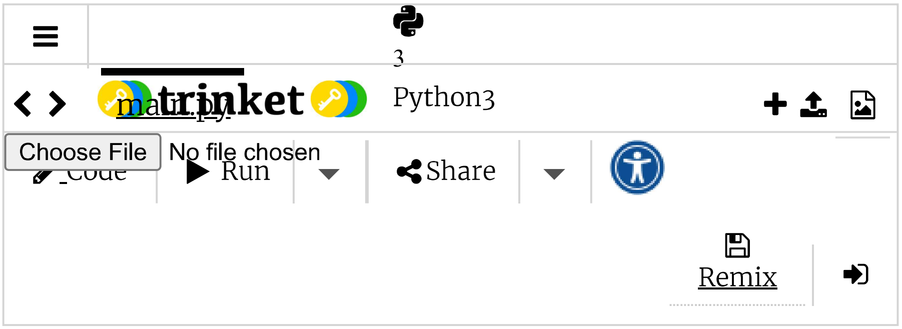
Check your response

Were your predictions correct? If not, how were they incorrect?

Nested Conditionals

There will be instances where you will want to test for more than one condition at a time. To do this, you can use what is called a nested `if` statement. This type of statement is an `if` statement placed inside another `if` statement.

4 Analyze the following program that has a nested `if` statement. Predict the output when you enter 12 and 75.



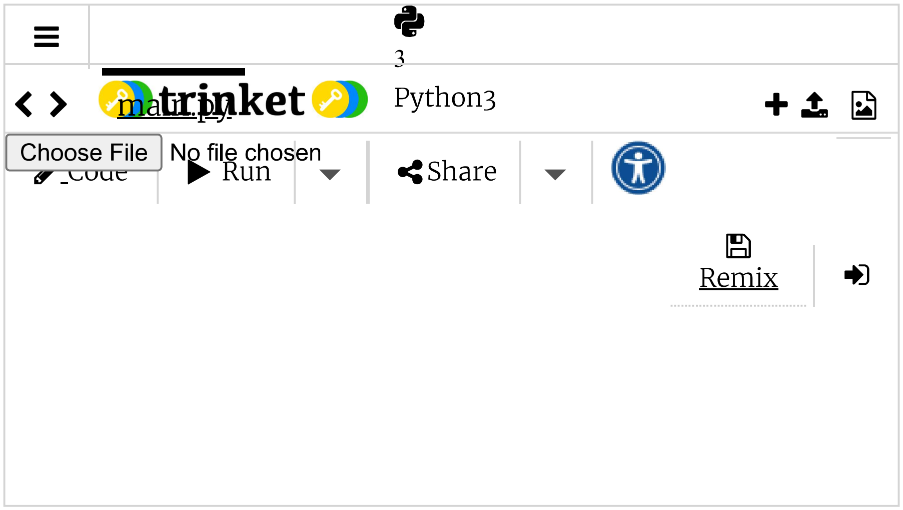
Was your prediction correct? If not, rerun the program a few times, varying the inputs.

5 In the above example, if a score is at least 60, a senior in high school may graduate. What if a senior could take summer school to meet graduation requirements? Add the following code segment to the above program in the correct place to produce “Summer school?” when a score is less than 60 for a high school senior.

```
else:
  print("Summer school?");
```

You can nest `if` statements many levels deep.

6 Analyze and predict the results of nested `if` statements three levels deep.

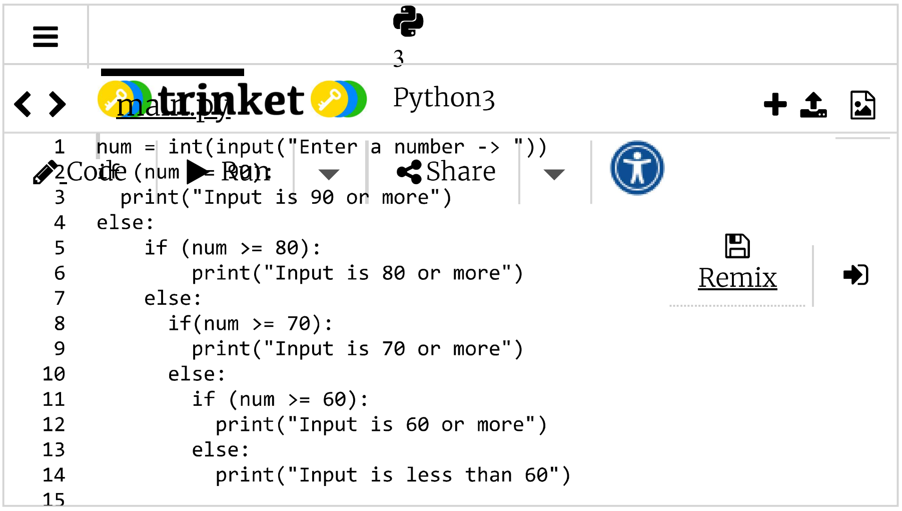


Were your predictions correct? If not, how were they incorrect?

Check your response

7 A new program contains nested statements four levels deep. Predict what will happen when the user enters different input

numbers.



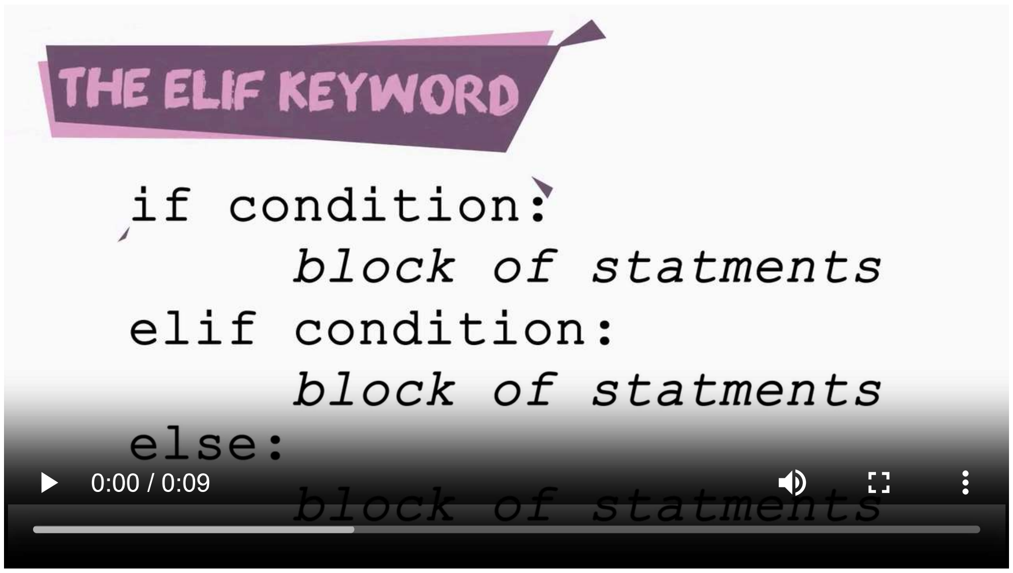
```
num = int(input("Enter a number -> "))
if (num >= 90):
  print("Input is 90 or more")
else:
    if (num >= 80):
        print("Input is 80 or more")
    else:
      if(num >= 70):
        print("Input is 70 or more")
      else:
        if (num >= 60):
          print("Input is 60 or more")
        else:
          print("Input is less than 60")
```

Were your predictions correct? If not, how were they incorrect?

Check your response

The `elif` Statement

As with most programming, you can accomplish goals in multiple ways. There is another, perhaps more straightforward, way to accomplish the same goals as the previous program. It is called the `elif` statement. (Notice it is a combination of `else` and `if`.)

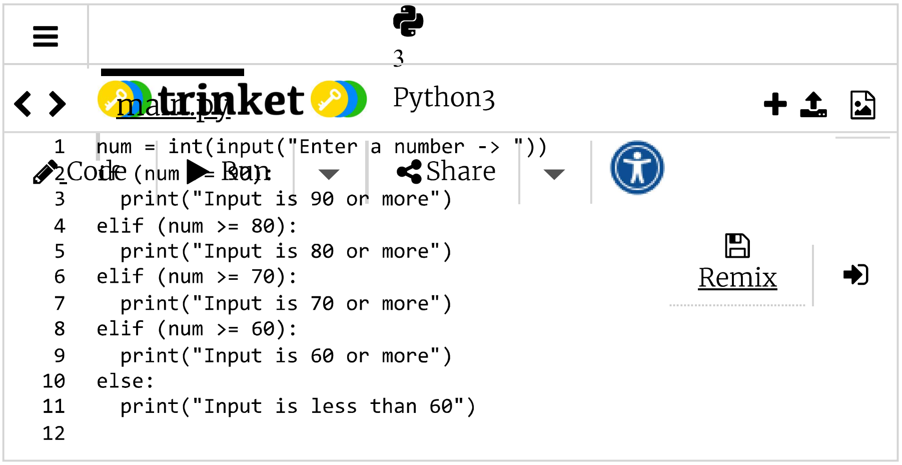


```
if condition:
        block of statments
elif condition:
        block of statments
else:
        block of statments
```

elif Statement Syntax video (The video has no sound.)

The elif statement will only execute after the previous conditions evaluate to false and its own condition evaluates to true. Explore an example of the elif statement in the following steps.

8 Examine the code. Compare it to the previous program. How do the two programs meet the same goal? Consider the length and complexity of the code.

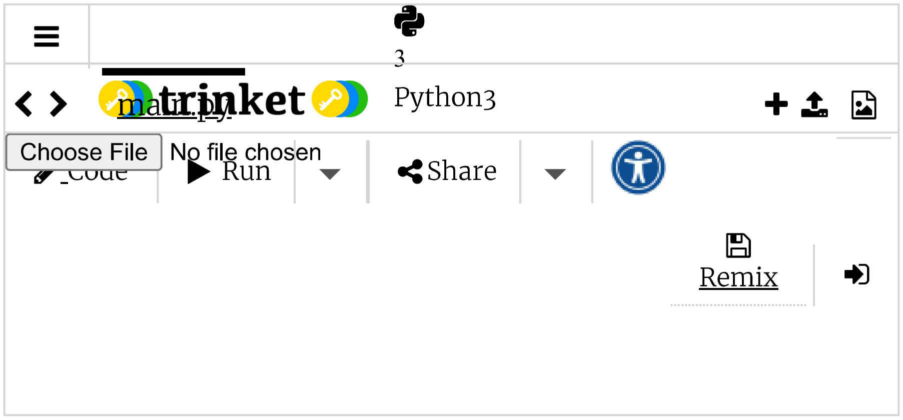


```
num = int(input("Enter a number -> "))
if (num >= 90):
  print("Input is 90 or more")
elif (num >= 80):
  print("Input is 80 or more")
elif (num >= 70):
  print("Input is 70 or more")
elif (num >= 60):
  print("Input is 60 or more")
else:
  print("Input is less than 60")
```

9 Revise the print statements in the code editor so that they display the letter grades A, B, C, D, and F. Then, change the values in the conditions to make it more or less difficult to earn a higher grade.

Comparing variables: Within a conditional statement, you are not limited to just comparing a variable with an integer. You can also compare variables to variables. For example, if the program above contained a variable `gradeA = 90`, you could write a conditional statement such as `if (num >= gradeA)`.

10 Sometimes, `if` statements can produce unexpected results. Within an `if` and `elif` structure, once one condition is matched, the rest will be ignored. Predict what will happen with a new, "reversed" variation.



11 Run the program a few times, trying to produce each letter grade. Did you predict the correct results?

Need Help?

Discussion Prompt: Discuss with a classmate why this algorithm produced the results that it did.

IMPORTANT: Take care in writing `if` statements. Be sure to test every part of the statement so that you see all parts of the `if/elif/else` statement execute correctly.

College Board Connection: The 'if' statement

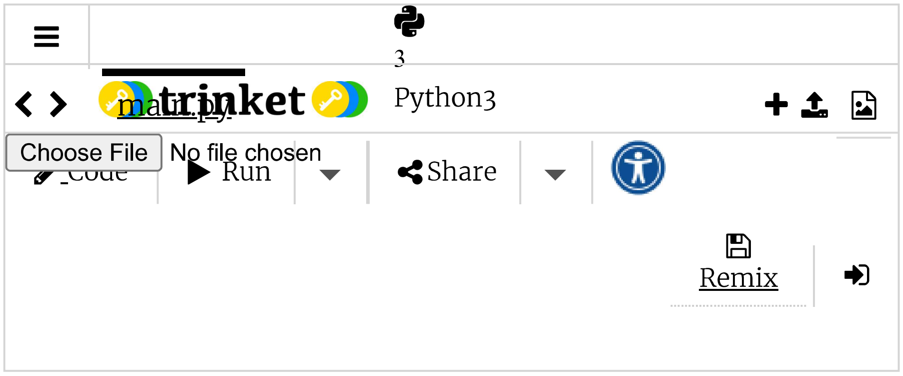
Nested `for` loop Iteration

Just like you can nest `if` and `while` statements, you can also nest `for` loop statements inside other statements. This is a simple but powerful construct in computer science. A `for` loop will execute a set of statements, once for each item in a list.

For example, observe the following pseudocode. The construct *`for i`* is the outer loop and *`for j`* is the inner loop.

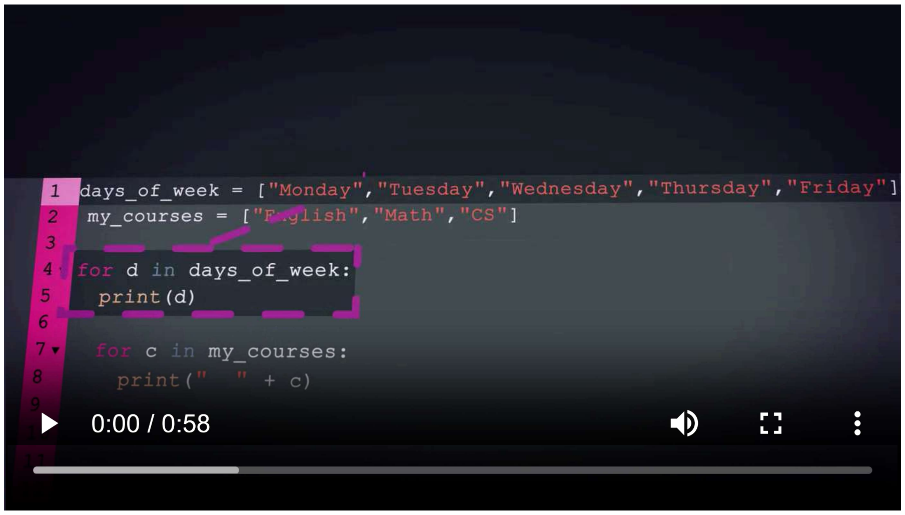
```
for i in some list
  for j in some list
    do something
```

12 Analyze the following code and predict what the output of these nested loops will be.



13 Run the code to see if your prediction was correct.

14 Explore how nested loops work. Use the video and “walk through” the algorithm, loop by loop:



```
days_of_week = ["Monday","Tuesday","Wednesday","Thursday","Friday"]
my_courses = ["English","Math","CS"]

for d in days_of_week:
  print(d)

  for c in my_courses:
    print("   " + c)
```

Nested Loops video (The video has no sound.)

- The `days_of_week` loop is called the outer loop, and the `my_courses` loop is called the inner loop.
- Begin with the outer loop. It iterates over the five `days_of_week` items, one at a time.
- On the first iteration, `d` is assigned Monday, as it's the first item in the `days_of_week` list. The loop prints Monday.
- Then the inner loop iterates over the three courses in the `my_courses` list, one course at a time. On this inner loop's

first iteration, `c` is assigned English, as it's the first item in the `my_courses` list. The loop then prints Monday.

- Each time the inner loop iterates, `c` changes to each course in the list (after English, `c` becomes Math, then CS). The outer loop does not iterate at this time, so `d` remains the same.
- When `my_courses` is done iterating, `days_of_the_week` now iterates for its second time and `d` becomes Tuesday, which it prints.
- The inner loop iterates over its three courses again, printing each corresponding item with each iteration. When it's done, the outer loop iterates to its next item in the list, so has a `d` value of Wednesday.
- This continues until the outer loop has iterated over the entire `days_of_week` list.

How many total iterations occurred, including iterations from both the inner and outer loops?

Check your response

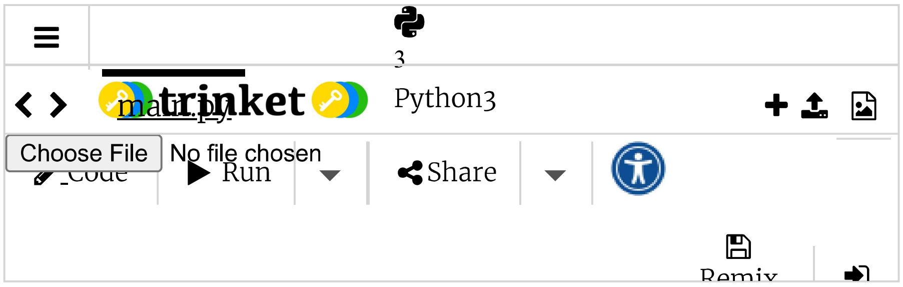
Outer and Inner Loops: Imagine if an outer loop had the state capitals and the inner loop had all street names in that city. The number of iterations with the same four lines of `for` loops could iterate *millions* of times. This is what makes nested loops a powerful computer science construct.

Nested Loops and Conditionals

Now you will create a more practical program using both the `my_courses` list and the grading algorithm. You will start with a small code segment and add to it so that the user can enter a grade for a variety of courses.

15 Create a new VS Code file called *a118_grades_[initals].py* and copy/paste this *incomplete* code segment into it. You will use some of the following steps to complete this code in VS Code.

Expected Error: The incomplete code will not display anything when run.

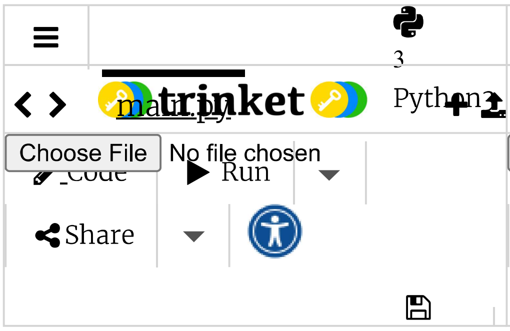


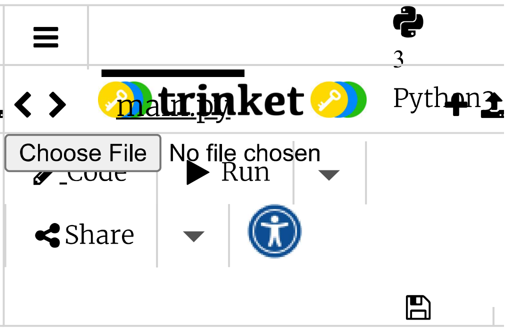
To help you plan your programs, you can express what you want to do using **pseudocode**. Pseudocode is a representation of what a program does (or what you want it to do), written in a way that most people can understand. For example, you will create the code to make the `grades` program perform the following tasks.

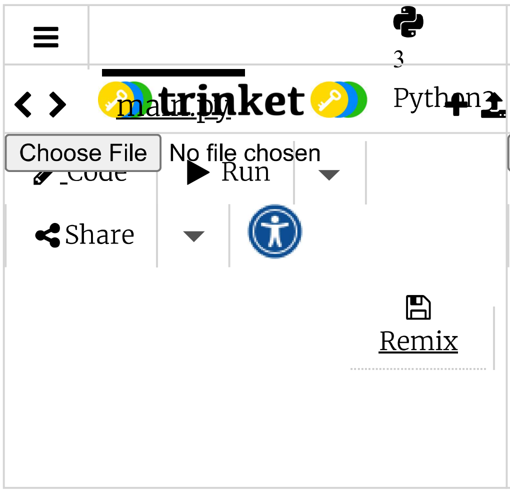
```
for each of the courses you are taking
    enter a point count
    display the corresponding letter grade based on point coun

if you want to redo the grades
    start over
otherwise
    program ends
```

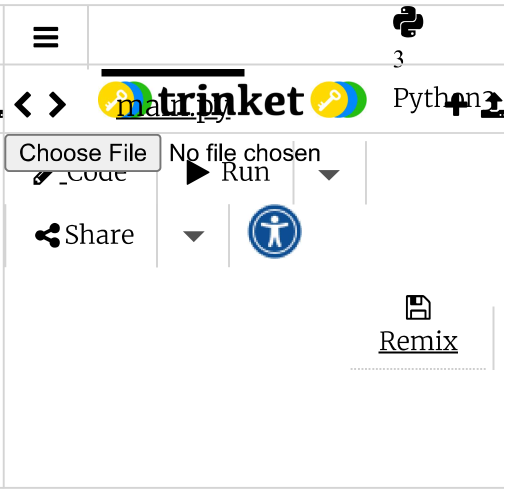
16 Use the four code segments below to complete the program described in the above pseudocode. Copy and paste these into your VS Code program.

> **Note**: The order and indentation of these incomplete segments are incorrect. To run properly, they need to be placed and indented properly in your program.









17 Test that you can produce each grade for at least one course, confirming that all parts of your program execute properly.

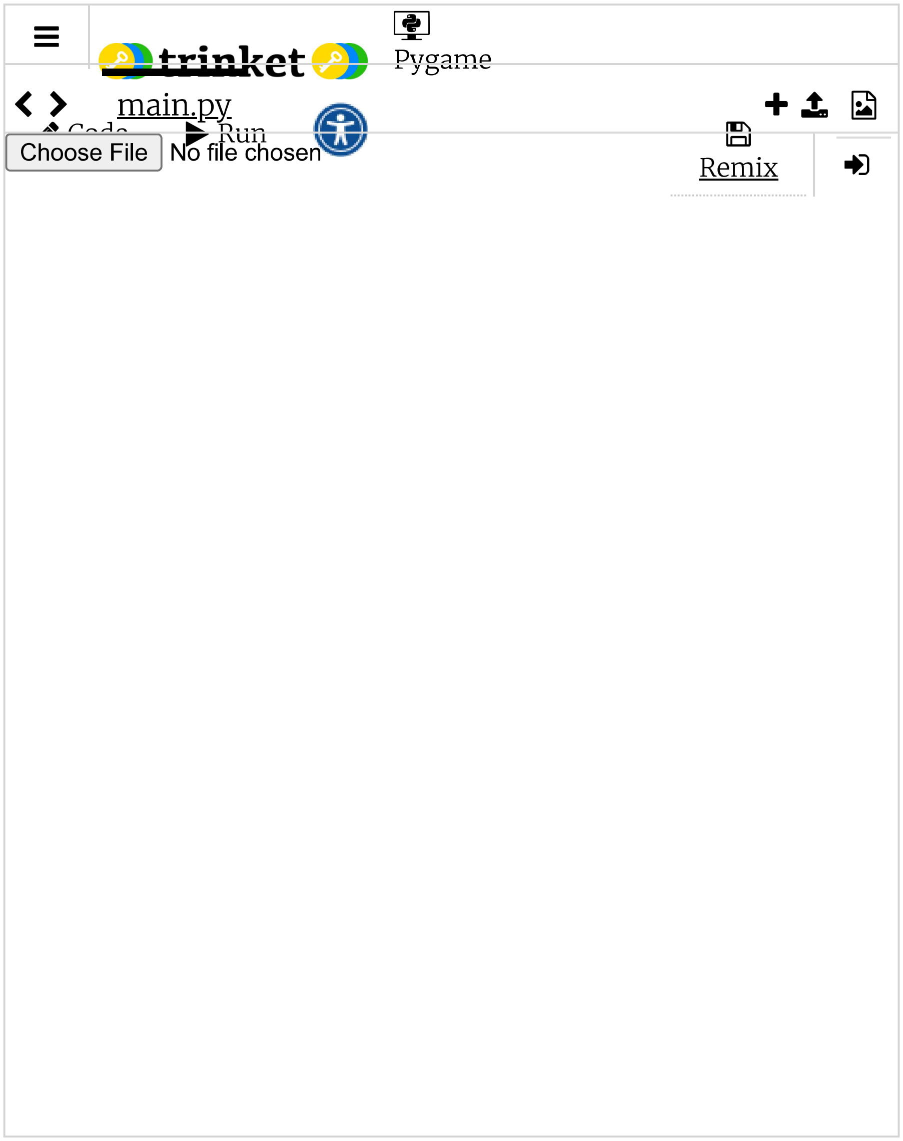
With this practice complete, you can now apply your knowledge of nesting, loops, and conditional statements to a bigger program.

Turtles Stop!

You will apply knowledge of conditionals and iteration to a turtle program. We have modified the code for Traversing Turtles (from Activity 1.1.6) so that it has two lists of turtles; one that places turtles on the left side of your screen, and one that places turtles across the top of the screen.

18 Copy and paste this code into VS Code as *a118_turtles_in_traffic_[initials].py*.

> **Note**: If you run this program in the code editor, you will not see the turtles. You need to run it in VS Code and use its screen to see them.



19 Review the code so that you understand how the turtle objects are created and placed on the screen. Add comments at the blank lines in the `for` loop, explaining what the code is doing. If necessary, run your program in VS Code to observe its behavior.

20 Your goal is to move the turtle objects across the screen (horizontally) or down the screen (vertically) and stop them if they collide. Read through the next five steps and write some pseudocode, planning how to accomplish this goal. Then,

begin writing your code.

a. You will process all of your turtles inside the `for step in range(50)` loop. Using nested loops, each turtle object takes fifty steps, either across the screen or down the screen.

b. To detect collisions, you need to know a turtle's x,y position on the screen. Recall the methods you used in the previous activity.

Method Name	**Example Call**	**Return value**
`xcor()`	`my_turtle.xcor()`	The current x coordinate of the turtle
`ycor()`	`my_turtle.ycor()`	The current y coordinate of the turtle

c. To detect collisions, you need to determine when turtles get too close to each other. In a coordinate system of positive and negative values, you need to get the *absolute value* of the distance between the two turtles. Use a new method `abs()` to calculate the distance between the turtle's x and y coordinates. For example, the statement `abs(x1 - x2)` determines the distance between a turtle's x position and `abs(y1 - y2)` determines the distance between a turtle's y position. If both distances are too close, there is a collision.

d. A good distance for turtles getting "too close" is about 20 pixels. Please note that the exact value varies depending on the specific shape of the turtle; you can ignore this detail for this activity.

e. If a collision does occur, stop moving the turtles that have collided. To do this, you can remove a collided turtle from its list. Recall the `pop()` method removes the last element in a list. *Python* provides another way to remove an element from a list: The `remove()` method searches for an element in a list and then removes it. You provide the element you want to be removed as a parameter to the method. For example, If you have a list of turtles, you can remove a turtle from the list with `remove(my_turtle)`.

Hints: Try to write the pseudocode and the solution yourself. Only use the hints below if absolutely necessary. This will help you prepare for enhancements and future projects and problems.

- [ ] See the pseudocode
- [ ] Solution code segments

It's Your Turn

You can make improvements to your `turtles_in_traffic` program to make it more interesting. For example, you can vary the speed of the turtles and/or you can change what happens when they collide. You may choose to implement all of the improvements described in the next two steps or only some of them. Check with your teacher to see if any are required.

21 Make faster and slower turtles:

a. Make the turtles speed up as they move. If they get "too fast," slow them down again. You can decide how quickly they speed up and when they need to slow down.

b. If you speed up the turtles, they cover more ground faster. Change your code so they do not go off screen.

22 Complete collision management:

a. Change the shape and color when the turtles collide: Delete one of your shapes in your list and use it as the collision shape. Use a new color to indicate a collision. The color does not have to be from your list of colors; you can choose any color as a collision color.

b. Try to recover from a collision. Make the turtles back up a bit and then continue on. You can still change the shape and color of collided turtles as an indicator that they crashed but also indicate they are able to recover. To recover from the collision, your code needs to remember and then restore the original shape and fill color of the turtle.

c. When the turtles stop moving and your program ends, you should indicate that this is intentional and that something did not “break” in the program. Write two `for` loops to change all of the turtles to some color that indicates they are deactivated.

CAREER CONNECTION

Graphics Programming

Turtle graphics can be a fun way to learn about programming. Knowing computer graphics and combining it with programming skills might make you interested in graphics programming. Animated movies and game graphics have graphics programmers as part of their creative team. A career as a graphics programmer requires an artistic or storytelling flair, attention to detail, a basic understanding of spatial math like geometry and trigonometry, and of course, good algorithmic programming skills.

Ifs and Nested Loops Check for Understanding

Get Started

3 Questions

AP CLASSROOM CONNECTION

Go to **AP Classroom** and follow your teacher's direction to review the Topic questions.

CONCLUSION

1 What did you find most challenging about this activity? What was the most enjoyable? Share your response with a classmate.

Proceed to project