Activity **3.2.3**

# Iteration and Counts

**Distance Learning Support**

Check with your teacher about:

☐ Materials or resources you need for this activity

☐ What work you need to turn in and how to submit it

☐ Collaboration strategies

## GOALS

- Learn to create, manipulate, and access lists in *Python*®
- Pair program to develop code that solves a problem and generates understanding

## TASKS DESCRIPTION

- Combo Menus Revisited: Modify your previously developed code to accept more than one order.

## ESSENTIAL QUESTION

1. How is iteration managing program flow?

## ESSENTIAL CONCEPTS

- *Python* Programming Language
- Coding Fundamentals: Algorithms, Functions, Variables, Arguments, Procedures, Operators, Data Types, Logic, Strings, and Concatenation
- Version Control, Iteration, and Debugging
- Tracer Bullet Development

## For Loops and Practicing Syntax

You've already seen **iteration** in the last activity in the form of a while loop. In *Python* you can use **`for` loops** and **`while` loops** to accomplish iteration. Anything that can be done with one type of loop, with some modification, can be done with the other.

One of the most common reasons to use a loop in *Python* is to perform some kind of action to every element in a list. In the last activity, you were given the `for` loop code to do this in *main.py*. For your reference, here is an example of iterating through a list.:

```
for post in all_posts_archive:
    print(post)
```

The first line contains:

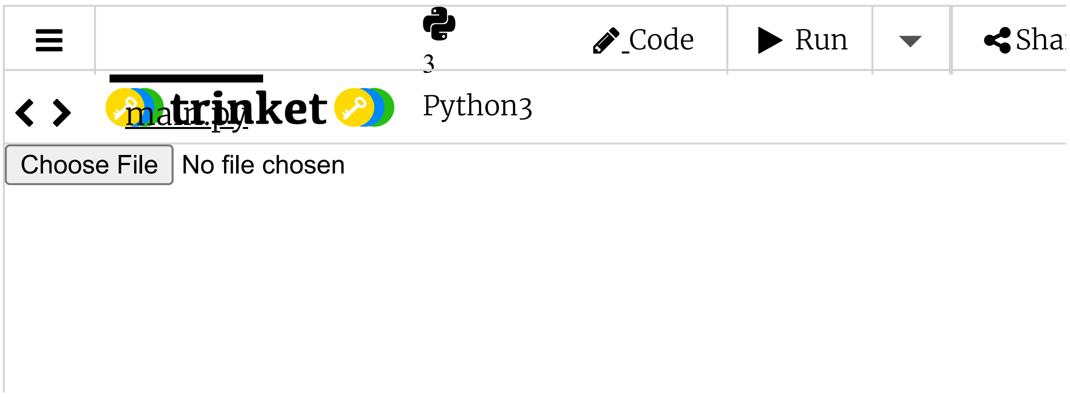
- the keyword `for`
- a variable identifier (in this case `post`) used to reference elements
- the keyword `in`
- a variable containing a list
- the `:` character showing that the rest of the loop will be on the next indented line

In other words:

| | |
|---|---|
| `for` | keyword |
| `post` | variable identifier of each individual element |
| `in` | keyword |
| `all_posts_archive` | a variable identifier for the list |

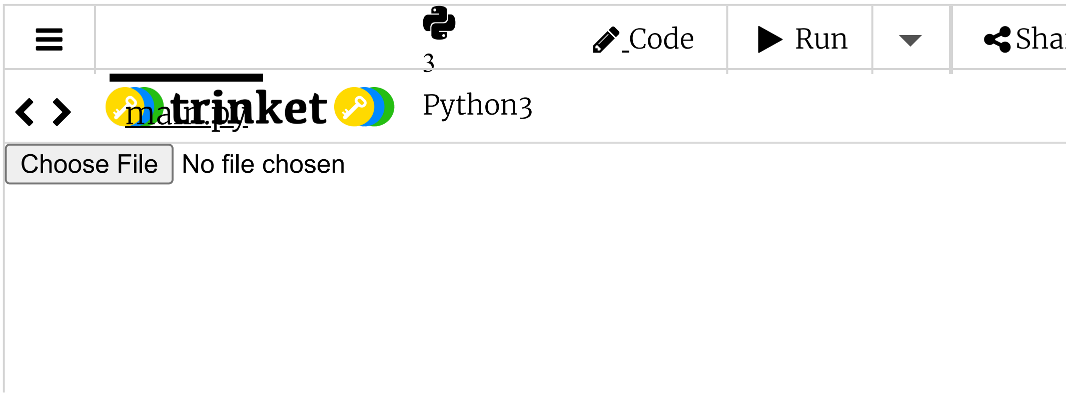
The way you would say this in natural language is "FOR each ELEMENT IN this LIST".

1 Examine the following loop. Before you run the code, predict what the output will be.



2. Run the code and see whether your prediction was correct.

3. Examine the following for loop. Predict what the output will be.



4. Test the code to see whether you are correct.

> **PLTW COMPUTER SCIENCE NOTEBOOK** - Loops TEMP Update
> Record the basic structure and syntax that defines a for loop.

5. Save a copy of your code as directed by your teacher.

## `range` and `list` Functions

Another useful function in *Python* is the **`range` function**. The `range` function identifies a number to start at, a number to stop at, and how much the number will change in each iteration. It loops through as many times as necessary to get to the end value. The syntax for using the `range` function is as follows.

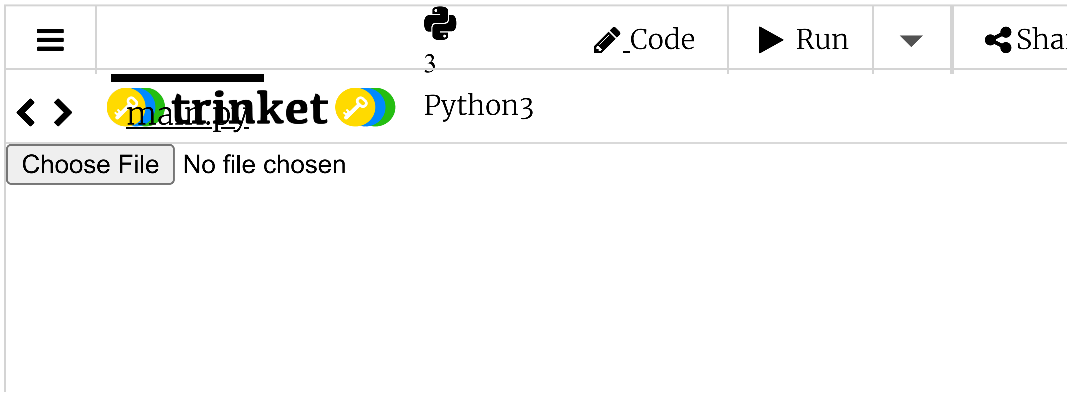
| Table Cell | Table Cell |
|---|---|
| `range(start, stop[, step]` | Returns a range from `start` to `stop`, incrementing by the value of `step`. If no `step` is provided, the default value is `1`. |
| `list(X)` | Converts `X` into a list of elements (in this case, from a range of numbers) |

```
range(start, stop[, step])
```

**Error Alert**: If the syntax is not exact, the loop will not execute.

**Optional arguments**: Notice the `[, step]` portion of the range function. This means that having this step is optional for the range function. You can choose to only use `range(start, stop)` and the function will work. To include a step, you will ignore the brackets and use `range(start, stop, step)`.

6 Review the following block of code.



**Error Alert**: The code will not run until `replace_with_numbers` is replaced with actual numbers.

7 Replace the argument `replace_with_numbers` with the arguments listed below. Try these examples one at a time, or by commenting out previous range functions after you have run them:

```
range(7)
range(1,8)
range(0,40,5)
range(0,16,3)
range(0,-10,-2)
range(0)
range(1,0)
```

**Note**: You can add comments that include the range and the output to help you track how this function works. Such as:

```
#range(5) output was [0, 1, 2, 3, 4]
```

8 Review the ranges in the code block and record your expectations for the outputs along with a brief explanation of why you think it will produce the lists you predict.

```
range(5)
range(2,5)
range(5,1,-1)
range(1,5,-1)
```

9 Test each `range` function to see whether you were correct.

**PLTW COMPUTER SCIENCE NOTEBOOK** - Range Function

Record notes for yourself about how the range function works. Especially note whether any `range()` behaved differently than you expected.

Save a copy of your code as directed by your teacher.

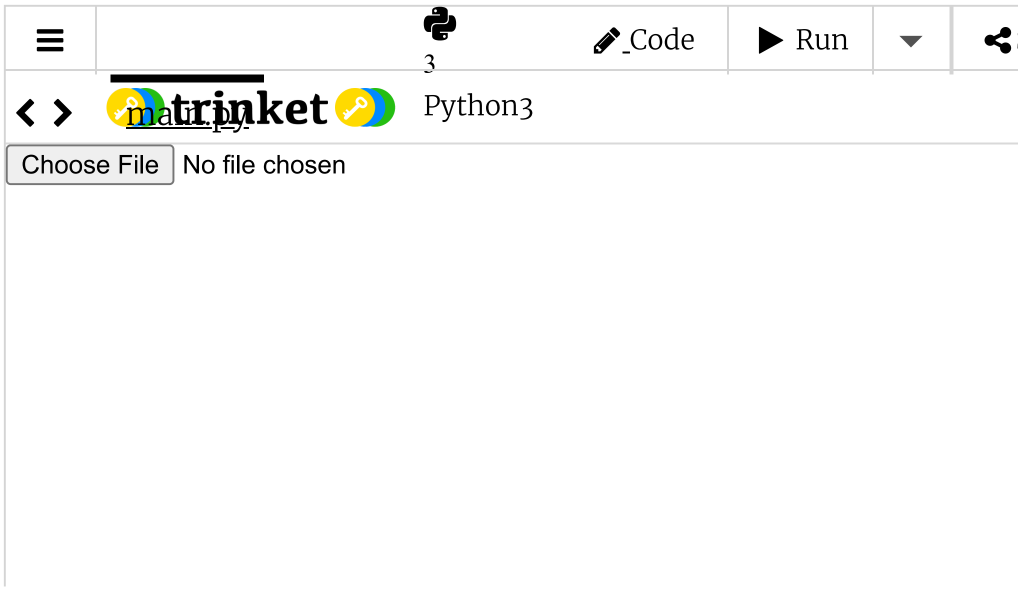
## Combining the `for` Loop and the `range` Function

In *Python*, `for` loops and the `range` function are often used together. Below is an example of a `for` loop that iterates 10 times. Notice how it might appear to iterate 11, but remember, 11 is not included in the range function as written.

```
for number in range(1,11):
    print (number)
```

10 Given the example above, try using the range function to write loops that print each of the following. Please note that "inclusive" means it should print all the numbers, including 1 and 19 in the first loop.

- ☐ Numbers 1 to 19 inclusive
- ☐ Numbers 20 to 1 inclusive
- ☐ Even numbers from 2 to 10 inclusive
- ☐ Odd numbers from 9 to 1 inclusive



**Check your code outputs**

**PLTW COMPUTER SCIENCE NOTEBOOK** - Ranges and Loops
Record information about using the `range` function in loops.

## `while` Loops

Another type of loop is a **`while` loop**. `while` loops check for a condition and continue to loop as long as the condition is true. The syntax of a `while` loop is shown below.

```
while (conditional expression with a variable):
  # what you want to happen
  # change the variable for the condition
```

In Activity 3.2.2, you were provided a while loop like the one below.

```
while (user_input != "quit"):
  # if/elif/else statements to check the user's decision
```

With `while` loops, the block of code is executed until the expression in the header evaluates to `False`. In the example of Activity 3.2.2, the header returned `True` (the user input does not equal `quit`) and looped the program. If the user entered `quit`, the evaluation would then return `False`. This allowed the user to continue entering commands.

Here is a `while` loop that prints the numbers from 0 to 10 inclusive:

```
x = 0
while (x < 11):
  print(x)
  x = x + 1
```

**PLTW COMPUTER SCIENCE NOTEBOOK** - While Loops

Record information about a `while` loop, such as what it is, what the syntax is, and when it might be used.

## Modulo Operator in Loops

The **modulo** operator is something you saw in block-based programming. It does division of two numbers, but it focuses on the remainder of the division instead of how many times the number is divided.

11 Take a minute to review modulo in block-based programming. What would this code section do, and how do you know?

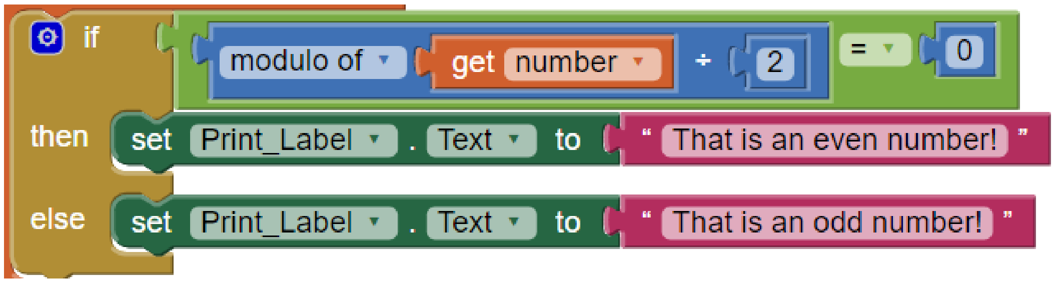

if
modulo of
get number
÷
2
=
0
then
set Print_Label . Text to
" That is an even number! "
else
set Print_Label . Text to
" That is an odd number! "

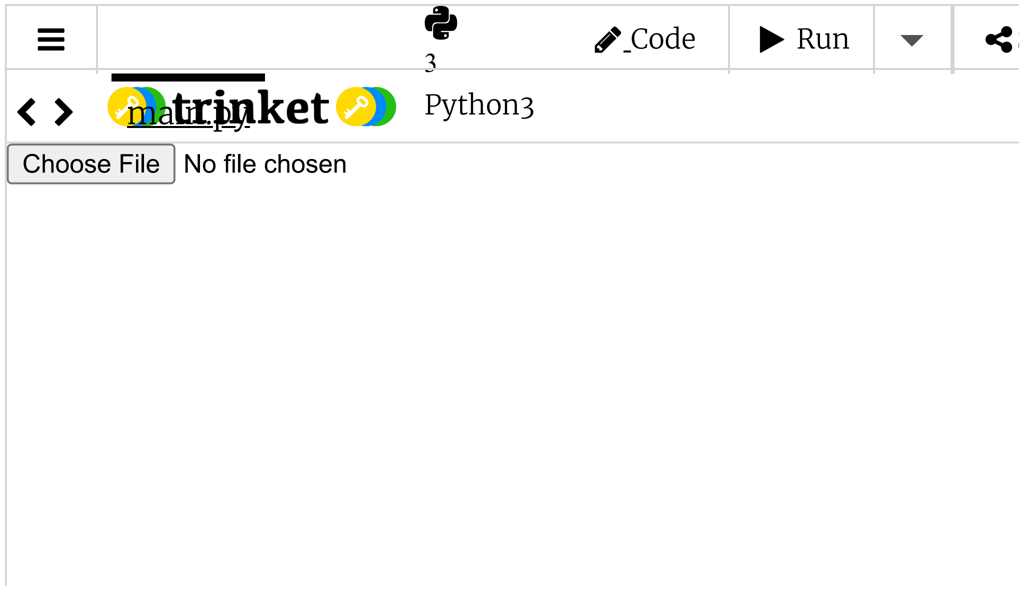
Figure 1. Example conditional using modulo in blocks.

Just like the MIT App Inventor example, you can use the modulo operator (`%`) in *Python* to determine evenness or oddness by finding the result of mod 2. If the result is 0, the number is even; if the result is 1, the number is odd. For example, the following expression evaluates to `True`.

```
4 % 2 == 0 # True, meaning 4 is an even number
```

12 Create and test `while` loops that print each of the following:

- ☐ Numbers from 1 up to 5 inclusive
- ☐ Numbers from 10 down to 1 inclusive
- ☐ Even numbers from 2 to 10 inclusive
- ☐ Odd numbers from 9 down to 1 inclusive; use the modulo operator and a conditional



**Important**: Having each of these `while` loops in the same file with the same variable will cause different results. Use different variable letters, comment out loops as you get the desired output, or use separate *Python* files.

**Check your code output**

## Combo Menu Iteration Using Loops

As you learned earlier, `for` loops are generally used to iterate over each element in a collection, like a list. They're also used to do something, a specific number of times.

`while` loops are generally used when execution may need to end early, or at a specific time, such as by user input as with the `Post` program.

13 Duplicate and rename your *combo_menu_list.py* to *combo_list_loop.py*.

14 Modify the Combo Menu code using the loop of your choice to make it so that the user has the option to keep entering more orders in the combo_list_loop file.

- It is possible to highlight multiple lines and press the Tab key on the keyboard to increase the indent of those lines.
- You can also unindent by highlighting lines and pressing **Shift**+**Tab**.
- You may need to change how the order is printed, but make sure it is printed inside the loop.

### CONCLUSION

1. Describe `for` loops and `while` loops.
2. How can the use of functions make your code more readable?
3. How do loops change the flow of control in a *Python* program?
4. How did you interpret and respond to the **Essential Question**? Capture your thoughts for future conversations.

Proceed to next activity