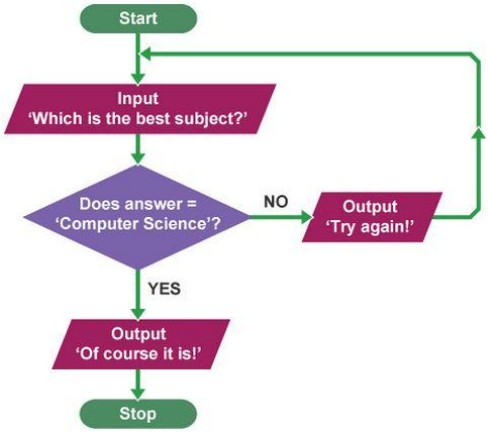
Let's take a moment to remind ourselves about what we are learning in this course and why.

If you can remember to the beginning of the course, basically we stated that this course is designed to show you how you can use computers **solve problems** and **make cool stuff**. We also said that *all* computer programming languages have a few key aspects:



1. **Input**
2. **Output**
3. **Decisions**
4. **Calculations**
5. **Repetition**

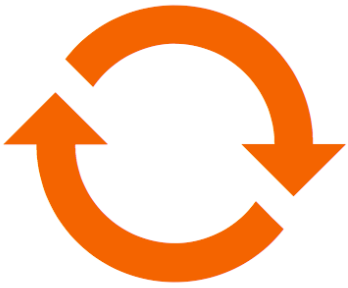
So far on our journey we have actually covered the *basics* of *most* of the stuff above! Good for you! Look at the words above. Can you do those things with python? You sure can!

We will now continue on our journey, adding to our tool kit, and strengthen your ability to **solve problems** and create **cool stuff** with computers!

Repetition (Looping)

Python For Loops

One of the most powerful features of computer programs is their ability to repeat actions automatically. In Python, there are several ways to achieve this, but the **for loop** is one of the most common and versatile. You might have already seen them (or used them) before. Here is how they work:



The `for i in range()` Function

The `range()` function is used to control how many times a loop repeats. You can pass between one and three integer **parameters** to it:

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

- **Start:** The integer value where the sequence begins (defaults to 0 if omitted).
- **Stop:** The integer the loop counts up to, but **does not include** (this parameter is always required).
- **Step:** The amount by which the value increases or decreases (defaults to 1 if omitted).

Examples of `range()` in Action

Exercise#1

Run the following for loops (individually by themselves) to see what they do

```
for i in range(6):
    print(i)

for i in range(20, 25):
    print (i)

for i in range(0, 15, 3):
    print (i)

for i in range(100, 50, -10):
    print (i)
```

Results for each in order:

Basic Loop (0 to 5)	0, 1, 2, 3, 4, 5
Custom range (20 to 24)	20, 21, 22, 23, 24
Using a Step (Up by 3s)	0, 3, 6, 9, 12
counting Backwards	100, 90, 80, 70, 60

Exercise#2

Using the `range()` function in python (and the examples above) to create **for** loops that will output each of the following sequences of numbers:

a) 0,1,2,3,4,5,6,7
b) 1,2,3,4,5,6,7
c) 2,3,4,5
d) 0,4,8,12,16,20
e) 10,15,20,25,30,35
f) 10,9,8,7,6,5,4,3
g) 1000,975,950,925,900,875

For loops with Lists and strings

In python, for loops have been designed to work easily with lists, strings and other data types. *Rather than* looping through a `range(),` you can simply define a list and loop through that list as shown below.

Exercise#3

Try the following Code

A. We'll assign a list to a **variable**, and then loop through the **list**:

```
sharks = ['hammerhead', 'great white', 'dogfish', 'frilled',
'bullhead', 'requiem']

for i in sharks:
    print(i)
```

```
Expected Output:
hammerhead
great white
dogfish
frilled
bullhead
requiem
```

B. Now with a **string**

```
word = "Mr. Walzl"
for i in word:
    print(i)
```

Exercise#4

Put into trinket

You can also use a **for** loop to **create** a **list:**

```
numbers = []

for i in range(10):
    numbers.append(i)      # { Remember:  .append  adds stuff to lists. }
print(numbers)

Output
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In the example above, the list `numbers` is initialized ***empty***, but the `for` loop populates the list.

Exercise#5

Using a for loop, and `.append(i)`, and `range()` to create a list named `up_by_fives[]` (This will be a list of numbers that goes up by fives from 0 to 50). Then print out the list.

Exercise#6

Here is a cool **function** that **counts the letters of any word** you give it. Type it in to Trinket, test it out, then save it or add it to your library of cool functions.

```
def count_letters(text):
    count = 0
    for c in text:
        if c == 'a':
            count = count + 1
    return count
print(count_letters("banana"))
```

Exercise#7

counting items in a list

Look at each of the 3 bit of code below. **Predict what the does**, *then* **enter the code into Trinket to see what it does:**

#7A

```
count=0
colors = ['red', 'orange', 'yellow', 'green', 'orange']
for x in colors:
  if x == 'orange':
     count = count + 1
print(count)
```

Flow control in FOR LOOPS

You can modify how a loop behaves using these two keywords:

- **break**: Immediately ends the loop.
- **continue**: *skips* the current step and moves directly to the next step in the loop.

break

Example of using "**break**" used in a for loop. A break is used to **end the loop** under certain conditions.

#7B

```
fruits = ['apple', 'orange', 'banana', 'cherry']
for x in fruits:
  if x == 'banana':
    break
  print(x)
```

continue

Example of using "continue" used in a for loop. Continue is used to **skip** a step of a loop under certain conditions.

"continue" will skip the block of code under certain conditions

#7C

```
fruits = ['apple', 'orange', 'banana', 'cherry']
for x in fruits:
  if x == 'banana':
    continue
  print(x)
```

Nested loops

A "Nested" for loop is a **for** loop inside another **for** loop.

In the example below each *store owner* gets printed 3 times with a second loop for *each fruit*.
See if you can **predict the output** for this code…then put it into Trinket.

Exercise#8

```
Store_owner = ['Tim','Sandy','Bill']
Fruits = ['apple','bananas','cherries']
for x in Store_owner:
  for y in fruits:
    print(x, y)
```

Exercise#8

Another "Nested" for loop.

A **For** loop *inside* another **for** loop.

In the example below 'Hey' gets printed **once** during the first pass of the loops, then twice on the 2nd pass through the loops, then 3 times on the 3rd pass through loops etc... See if you can **predict the output for this code** will look like...**then put it into Trinket.**

```
for i in range(7):
    for j in range(i):
        print('Hey')
    print('')
```

Exercise#10 Test your Understanding of for loops:

a) Write a program which sums the integers from 1 to 10 using a for loop (and prints the total at the end).

b) Write a program which finds the **factorial** of a given number that is input by the user. Examples: 3 factorial, or 3! is equal to 3 x 2 x 1;
5 factorial, or 5! is equal to 5 x 4 x 3 x 2 x 1,

Your program should only contain a single **for** loop.

c) Write a Python program to find those numbers which are evenly divisible by 7 and evenly divisible by 5 (between 1500 and 2700). Use a for loop with a range(). This code should be less than 10 lines.

d) **Vowel Counter:** Create a program that uses a for loop to count the vowels in a word entered by the user.

🚀 Level Up: More fun Python For Loop Challenges

Exercise #11: The Square Maker

Using a `for` loop and the `.append()` method, create a program that calculates the "**square**" (x^2) of every number from 1 to 10.

- **Goal**: Create a list called `squares[]`.
- **Action**: Loop through `range(1, 11)`, calculate the square of each number, and append it to your list.
- **Result**: Print the final list. It should look like: `[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]`.

Exercise #12: The "No-Space" Zone

In Exercise #6, you counted letters in a word. Now, let's use the `continue` keyword to filter a string.

- **Goal**: Ask the user to input a full sentence.
- **Action**: Loop through the string. If the character is a **space** (" "), use `continue` to skip it. Otherwise, print the character.
- **Result**: Your program should print the user's sentence back to them, but with all the spaces removed!

Exercise #13: The Multiplier Table

We know computers are great at calculations. Let's make a simple multiplication tool.

- **Goal**: Ask the user for a number (e.g., 7).
- **Action**: Use `for i in range(1, 11)` to print the multiplication table for that number.
- **Result**: The output should look like:
  - 7 x 1 = 7
  - 7 x 2 = 14 ... and so on up to 10.

Exercise #14: Finding "The One" (Search and Break)

Sometimes you don't need to finish a loop if you've already found what you're looking for.

- **Goal**: Create a list of 10 random numbers.
- **Action**: Loop through the list to find the first number that is **greater than 50**.
- **Result**: When you find it, print "Found a big number: [the number]" and use `break` to stop the loop immediately.

Exercise #15: The Star Grid (Nested Logic)

Building on your nested loop practice, let's create a visual pattern.

- **Goal**: Use a nested `for` loop to print a 5x5 grid of stars (*).
- **Action**: The outer loop should run 5 times (for the rows), and the inner loop should run 5 times (for the stars in each row).
- **Hint**: Use `print('*', end=' ')` in your inner loop to keep the stars on the same line, and a simple `print('')` in your outer loop to move to the next row.