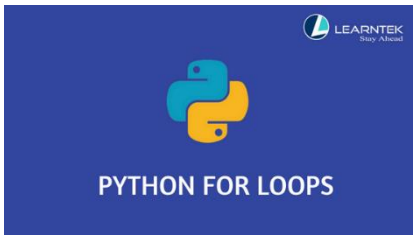
For loops in Python



One of the *fantastic* things that computer programs do is **repeat things**. This can be done a few ways in Python:

1. **While** loops (which you should already be familiar with)
2. **Iteration** (advanced technique involving a function that calls itself) – wait till next year.
3. **For loops**

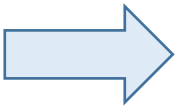
For loops are **VERY** common in python. You might have already seen them (or used them) before. Here is how they work:

Most commonly for loops are used for looping over a sequence of data (a list, characters in a string etc).

Examples:

For Loops using `range()`

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



```
Output
0
1
2
3
4
5
```

will loop through code 6 times.

`range()` is used to control how many times the loop will be repeated.

When working with `range()`, you can pass between 1 and 3 integer ***parameters*** to it:

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

- `start` states the integer value at which the sequence begins. If this is not included then `start` begins at 0
- `stop` is **always required** and is the integer that is counted up to **but not included**
- `step` sets how much to increase (or decrease in the case of negative numbers). If step is omitted then `step` defaults to 1

We'll look at some examples of using the different ***parameters*** in `range()`.

If you only put one number into `range()` it becomes the `stop` parameter:

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

```
Output
0
1
2
3
4
5
```

Now we will try `range(start, stop)`

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

```
Output
20
21
22
23
24
```

`range(start, stop, step).`

Let's cout up by 3's from 0 to 15 (but of course not including 15)

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

```
Output
0
3
6
9
12
```

We can also use a negative value for our `step` argument to iterate **backwards**.

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

```
Output
100
90
80
70
60
```

Exercise#1

Using the `range()` function in python (and the examples above) 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 using Lists or strings

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

Example:

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(shark)
```

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

You can also use a `for` loop to construct a list from scratch:

```
numbers = []

for i in range(10):
    numbers.append(i)
print(integers)
```

{**Remember:** `.append` adds stuff to lists.}

```
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#2

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

For loops work great with **strings** in python too:

```
sammy = 'Sammy'

for letter in sammy:
    print(letter)
```

```
Output:
S
a
m
m
y
```

Exercise#3

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#4

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

a)

```
count=0
colors = ['red', 'orange', 'yellow', 'green', 'orange']

for x in colors:
  if x == 'orange':
     count = count + 1
print(count)
```

break

b) Example of using **"break"** used in a for loop. A break is used to **end the loop** when needed.

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

continue

b) Example of using **"continue"** used in a for loop. Continue is used to **step out of a loop only once and then continue** when needed.

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

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

Advanced Stuff: Nested loops

c) A "Nested" for loop. 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.

```
Store_owner = ['Tim','Sandy','Bill']
Fruits = ['apple','bananas','cherries']

for x in Store_owner:
  for y in fruits:
    print(x, y)
```

d) 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#5

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) Write a python program that uses a `for` loop to find out how many vowels are in a word that is entered by the user.