

Unit 4: For Loops and Lists

Skill Builder 3: Throwing in Some Randomness

In this lesson, you will make a list of random numbers to investigate and analyze patterns.

**Objectives:**

- Use `randint(a, b)`
- Create a list of random integers
- Analyze properties of the list

**Teacher Tip:** This lesson can be tricky. Syntax errors may pop up, and Python error messages are sometimes not sufficient to debug the program. There is a suggested extension project at the end of the lesson which can be a challenge. Omitting it will not interrupt the lesson flow.

**Background**

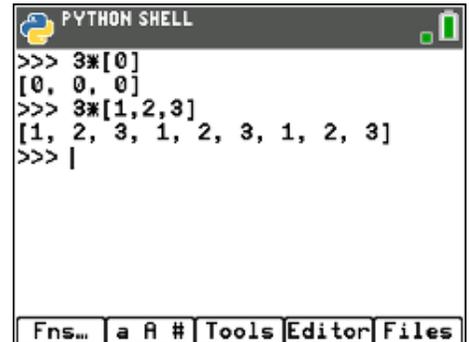
Lists (or arrays) are a convenient way to store many different values in a single variable. Python is very flexible with the contents of a list (it can hold different types of values, but this is rare) and has several ways of making a list.

One expression worth noting is `3 * [0]` which “replicates” the element three times.

What does `3 * [1,2,3]` produce? The same expression in the calculator app gives a completely different result as shown in the second image.

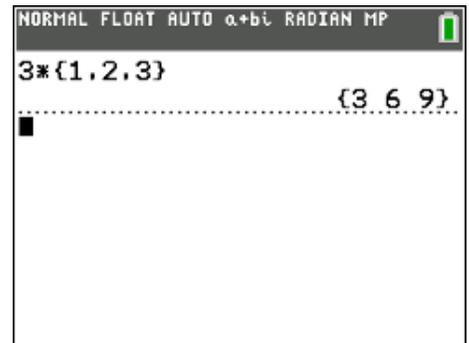
Use caution when working with lists in Python because the result might not be what you expect.

We will stick with the basics in this lesson which makes a list of random numbers and then calculates some “1-Var Statistics” on the data.



```

PYTHON SHELL
>>> 3*[0]
[0, 0, 0]
>>> 3*[1,2,3]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> |
    
```



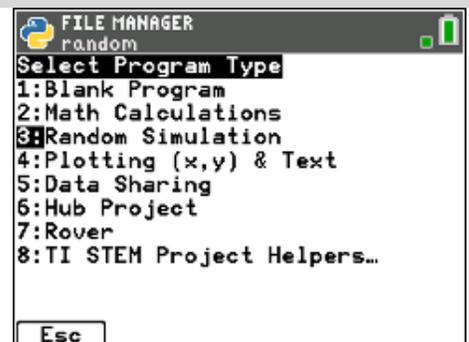
```

NORMAL FLOAT AUTO a+bi RADIAN MP
3*[1,2,3]
----- {3 6 9}
    
```

**Teacher Tip:** Check online documentation for all the different ways of creating and using Python lists. Use `print()` statements often to see what’s going on with the variables. It’s easy to print a list: `print(listname)`.

**Lists** are one kind of Python data structure. Other similar structures are tuples, sets and dictionaries, each with their own features and restrictions. The others are not included in this course.

1. For this project, when creating the **<New>** Python program, when entering the program name, choose **Random Simulation** from the **<Type>** menu.



```

FILE MANAGER
random
Select Program Type
1:Blank Program
2:Math Calculations
3:Random Simulation
4:Plotting (x,y) & Text
5:Data Sharing
6:Hub Project
7:Rover
8:TI STEM Project Helpers...
Esc
    
```

- Our program is named RANDOM1 and the template provides a comment and the statement:

**from random import \***



```
EDITOR: RANDOM1
PROGRAM LINE 0003
# Random Simulation
from random import *
-
```

- Do not select anything yet, but look at the contents of the **random** module:

Select <Fns...> **Modul** > **random...** to see that it contains several functions that generate “random” values. **randint(min, max)** generates a single random number between *min* and *max*, values that the programmer provides.



```
EDITOR: RANDOM1
random module
random
1:from random import *
2:random()
3:uniform(min,max)
4:randint(min,max)
5:choice(sequence)
6:randrange(start,stop,step)
7:seed()
Esc | Modul
```

Select <Esc>.

**Teacher Tip:** Computers use sophisticated algorithms to generate “pseudo-random” numbers. But the methods are predictable: Providing the same value for the **seed()** function included in the random module will produce the same sequence of random numbers every time the program is run. This is useful for testing but not a good idea when trying to simulate true random events. There are techniques (many using an internal clock) for seeding the random numbers with a random number to give a better *feeling* of randomness.

- In the Editor, first create an empty list:

**nums = [ ]** (with nothing between the brackets)

Use a **for** loop to make a list of 100 random numbers, each selected from the values between and including 1 and 25. *You* must enter the **range()** size 100.

The loop *block* is:

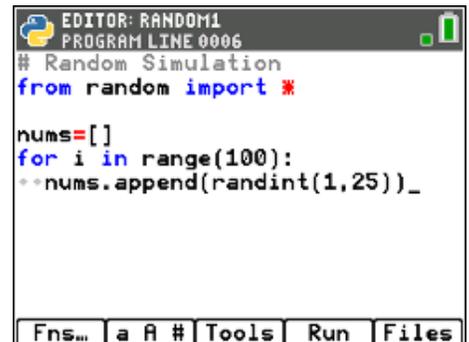
◆◆ **nums.append( randint(1,25) )**

Select <Fns...> **List** for **.append( )**

With your cursor inside the **.append( | )** parentheses...

Select <Fns...> **Modul random... randint( )**

and add the values **1 , 25** for the two arguments.



```
EDITOR: RANDOM1
PROGRAM LINE 0006
# Random Simulation
from random import *
nums=[]
for i in range(100):
    nums.append(randint(1,25))_
```

Pay attention to the *two* right parentheses at the end of this statement, a common syntax error if you are just typing code. If you use the menus the parentheses are provided.



**Teacher Tip:** The TI-84 Plus CE's OS has an optional third argument: `RandInt(min,max,n)` which causes the function to generate a list of numbers between min and max, but Python has no such option.

Another slick method uses Python "list comprehension":

```
nums = [randint(1,25) for i in range(100)]
```

5. Add a `print( )` statement to the program (after the `for` loop) to print the list after all the numbers have been created:

```
print(nums)
```

*Note that this statement is dedented.*

Run the program now to make sure it works. Re-running the program in the Shell will display a different set of numbers in each run. To quickly re-run a program:

- Select **<Tools>** then press **[enter]** or
- Select **<Editor>** then select **<Run>** by pressing **[trace]** twice.

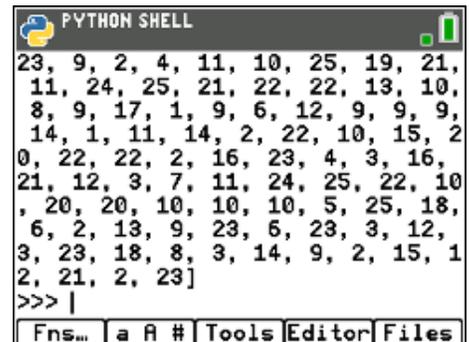


6. *Here's a sample run.*

Add code to your program to determine the average of the numbers. Try it yourself first. (Remember the last lesson!)

Display the minimum and maximum values in the list. (Hint: See **<Fns...> Lists**.)

*Can you sort the elements? Check the **Lists** menu again!*



7. **About sorting**

Python has two tools for sorting a list:

`nums.sort()` arranges the elements of `nums` into *ascending* order:

```
print(nums)           unsorted
```

```
nums.sort()
```

```
print(nums)           sorted
```

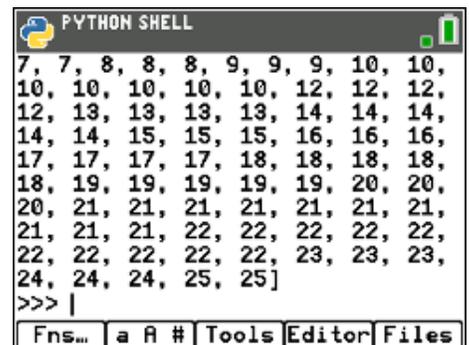
*(changes the list nums)*

`sorted(nums)` returns a list that is sorted but *does not change the original list*. Use `nums2 = sorted(nums)` to keep the original list and make a new, sorted list named `nums2`.

```
print(nums)           unsorted
```

```
nums2 = sorted(nums)
```

```
print(nums2)         sorted
```





*(does not change the list nums)*

**Teacher Tip:** `list.sort()` and `sorted(list)` are two different methods of the list class.

`list.sort()` changes the list and returns **None**.

`newlist = sorted(list)` does not change the list but *returns* a sorted list.

Note the difference in syntax. Check online for more information.

#### 8. Extension

After sorting the list, it is now possible to determine the **median** (middle) value and the **range** (difference between minimum and maximum) of the data set.

Add statements to your program to display these values.

*Note: If the length (size) of the list is an even number, then the median is the average of the two "middle" numbers.*

Use the `len(<list>)` function found on **<Fns...> Lists** to determine the length (size) of the list.

Can you also determine the Q1(first quartile) and Q3 (third quartile) values?

```

PYTHON SHELL
6, 6, 7, 7, 8, 8, 8, 8, 8, 9, 9
, 9, 9, 10, 10, 10, 10, 11, 11,
11, 12, 12, 12, 12, 12, 13, 13,
13, 13, 13, 14, 14, 14, 14, 14,
14, 15, 15, 15, 15, 16, 16, 16,
16, 17, 17, 17, 17, 18, 19, 19,
20, 20, 20, 21, 21, 21, 21, 22,
22, 23, 23, 23, 23, 23, 24, 24,
24]
median: 11
>>> |
Fns... | a R # | Tools | Editor | Files

```

**Teacher Tip:** Solution for the optional exercise (being careful not to use reserved words!):

```

lenth = len(nums2)
if lenth % 2 == 0:
    medi = (nums2[lenth // 2] + nums2[lenth // 2 + 1]) / 2
else
    medi = (nums2[lenth // 2])
print("Median=",medi)
rang = max(nums2) - min(nums2)
print("Range = ",rang)

```

Extra challenge: Determine the **mode(s)**!