

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 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 the help messages are not very helpful. There is a suggested extra project at the end of the lesson which can be a challenge so omitting it will not interrupt the lesson flow.

Introduction:

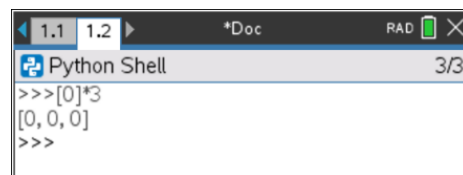
Lists (or arrays) are a convenient way to store many different values in a single variable. Python is very flexible with the contents of a list (it can hold different types of values, but this is rare) and has several ways of making a list.

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

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

Use caution when working with lists in Python because the result might not be what you are accustomed to!

We will stick with the basics in this lesson...



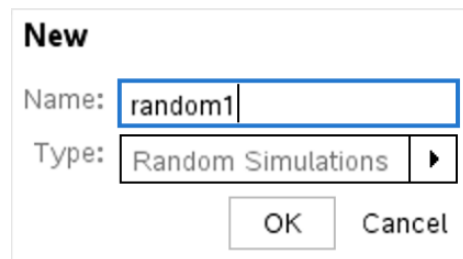
```
>>>[0]*3
[0, 0, 0]
>>>
```



```
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 is really easy to print a list: **print(listname)**.

1. For this project, when creating the new Python program, choose 'Random Simulations' from the **Type:** dropdown list.



New

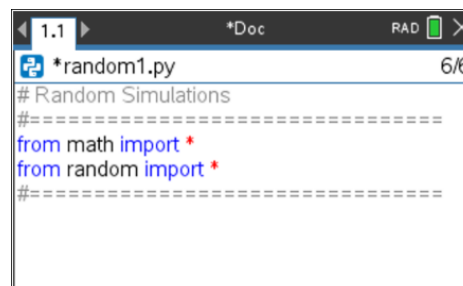
Name:

Type: 

2. The *Random Simulations* template provides you with two modules: **math** and **random**. You've seen that **math** includes functions like `sqrt()`, `trig` functions, `exp()` and `log()`, and others.

The **random** module contains functions that work with random numbers.

One useful function is **randint(a, b)** which returns a **random integer** between and including the values of **a** and **b**. The arguments can be numbers, variables or expressions that yield a number.



```
*random1.py
# Random Simulations
#=====
from math import *
from random import *
#=====
```



Teacher Tip: Computers use sophisticated algorithms to generate 'pseudo-random' numbers. 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.

3. 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 range 0 to 25 (you supply the *index* variable and range size).

The loop *block* is:

nums.append(randint(1,25))

Pay attention to the two right parentheses at the end of this statement: a common syntax error.

Teacher Tip: Another method using 'list comprehension':

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

4. Add a print statement to the program (after the **for** loop) to print the list after the numbers have been created:

print(nums)

Run the program now to make sure it works. Pressing **ctrl+R** in the Shell will re-run the program and you will see a different set of numbers in each run.

5. 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 **menu > Built-ins > Lists.**)

Can you sort the elements?

```

1.1 1.2 *Doc RAD 9/9
# Random Simulations
#=====
from math import *
from random import *
#=====
nums = []
for index in range(size):
    nums.append(randint(1,25))

```

```

1.1 1.2 *Doc RAD 9/9
# Random Simulations
#=====
from math import *
from random import *
#=====
nums = []
for i in range(100):
    nums.append(randint(1,25))
print(nums)

```

```

1.1 1.2 *Doc RAD 83/83
Python Shell
[9, 17, 7, 18, 2, 6, 5, 8, 6, 6]
>>>#Running random1.py
>>>from random1 import *
[4, 14, 24, 12, 6, 17, 1, 14, 14, 20, 8, 3, 4, 18, 20,
4, 24, 9, 2, 15, 3, 10, 22, 19, 22, 6, 10, 13, 6, 3, 1
9, 24, 22, 3, 24, 23, 22, 6, 17, 19, 1, 4, 21, 5, 8, 2
4, 4, 19, 22, 18, 19, 4, 15, 1, 24, 23, 10, 10, 20, 2
3, 17, 13, 17, 7, 1, 17, 11, 23, 25, 23, 3, 14, 25, 2
5, 12, 15, 3, 10, 4, 16, 13, 20, 1, 8, 20, 8, 25, 23,
10, 5, 9, 25, 10, 21, 7, 23, 6, 13, 3, 4]
>>>

```



6. About sorting...

Python has two tools for sorting a list:

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

sorted(n) returns a list that is sorted but *does not change the original list*.

Use **nums2 = sorted(n)** to keep the original list and make a new, sorted list named **nums2**.

```

1.1 1.2 *Doc RAD
Python Shell 101/101
22, 23, 23, 23, 23, 24, 24, 24, 25]
>>>#Running random1.py
>>>from random1 import *
[1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 12, 12, 13, 13, 13, 13, 14, 14, 15, 16, 16, 16, 16, 16, 17, 18, 18, 18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20, 20, 21, 21, 22, 22, 22, 22, 23, 23, 24, 24, 24, 24, 24, 25, 25, 25, 25, 25]
>>>

```

Teacher Tip:

list.**sort()** and **sorted(list)** are two different methods of the list class.

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

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

Note the two different syntaxes. Check online for more information.

An optional exercise:

7. 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 **menu > Built-ins > Lists** to get the length of the list.

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

```

1.1 1.2 *U4SB3 r...ode RAD
Python Shell 29/29
[1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 6, 6, 6, 6, 7, 7, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12, 12, 12, 13, 13, 14, 14, 14, 14, 14, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 17, 17, 18, 18, 18, 18, 19, 19, 19, 19, 20, 20, 20, 21, 21, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 24, 24, 24, 24, 24, 25]
>>>
Median= 14.0
Range = 24
>>>

```

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)**!