ICS3U: Random Values

Random Numbers

Many video games use randomness to simulate rolling dice, dealing cards, or spawning enemies. Python has the ability to generate pseudo-random values, which are close enough for what we will want to do in this course, and to manipulate data in randomized ways.

All of this functionality is found in the `random` module. Like the `math` module, you must remember to `import` the `random` module or none of its functions will be available to use. A run-time error will occur when attempting to use a function. As with any module, it is necessary to prefix any function call with `random`, as in `random.function`.

The `random` module provides several functions for generating random integers, including `randint` and `randrange`. The first, `randint`, takes two arguments for the lower and upper bounds. These values are both included in the random selection. The code below simulates rolling two six-sided dice and calculating their sum.

```
import random
die1 = random.randint(1, 6)
die2 = random.randint(1, 6)
dice_sum = die1 + die2
print("You rolled a", die1, "and a", die2, "for a sum of", dice_sum)
```

A similar function, `randrange`, allows for a third argument that acts as a **step value** (i.e. what to count by) in the range of values. Unlike `randint`, the upper bound for `randrange` is not included in the selection. This is to provide functionality to other similarly-named functions, such as `range`, which we will encounter shortly. The code below will generate two random values, one even and one odd, between 10 and 30. To ensure that each value was even or odd, a step value of 2 was used to avoid intermediate values. Note that for 30 to be included, the upper bound for the even value needed to be specified as 31.

```
import random
random_even = random.randrange(10, 31, 2)
random_odd = random.randrange(11, 30, 2)
print(random_even, "is even and", random_odd, "is odd")
```

Generating a random real number is a little more involved. One of many functions that deal with real numbers is `random`. This function generates a random value between 0 and 0.999…. It shares the same name as the module itself, and takes no arguments to specify lower or upper bounds. In order to generate values outside of this range, it may be necessary to multiply the randomly-generated value by some constant (to adjust the width of the range) or add or subtract some constant (to adjust the starting point).

Let's generate a random real number between 10 and 40. The range between 10 and 40 is 30, so we must scale our random value by a factor of this amount. This will create a real number between 0 and 30. Next, we increase the random value by 10, so that it aligns with the starting point of our range. Run the code on the next page several times to convince yourself that it works as intended.

```
import random
rand_value = random.random()
rand_value *= 30
rand_value += 10
print("A random real number between 10 and 40 is", rand_value)
```

The same the can be done more concisely by combining these operations into a single line.

```
import random
rand_value = 30 * random.random() + 10
print("A random real number between 10 and 40 is", rand_value)
```

Random Selections from Sequences

Making random selections from a sequence, such as a string, can be done using the `choice` function provided by the `random` module. This function takes a sequence as an argument and selects one element from the sequence. In the case of a string, this corresponds to a single character.

```
import random
letters = "ABCDE"
random_letter = random.choice(letters)
print("A random letter from A-E is", random_letter)
```

While it is usually considered good programming practice to define your random options in a variable like `letters`, it is also possible to use a literal string as the argument for `choice`.

```
import random
random_letter = random.choice("ABCDE")
print("A random letter from A-E is", random_letter)
```

So far, the only sequence data types that we have really dealt with are strings, but choice can handle any sequence.

A *loaded die* (or *weighted die*) is one that is manufactured or modified so that it is more likely to land on certain values than others. Suppose we wanted to simulate a loaded die that rolled 6 50% of the time, and the other five values 10% of the time each. We can specify a sequence of ten integers where the value 6 occurs five times, and the other values each occur once. Such a program might look something like this.

```
import random
loaded_die = (1, 2, 3, 4, 5, 6, 6, 6, 6, 6)
roll = random.choice(loaded_die)
print("You rolled a", roll)
```

Try running the code above 20-30 times (or, ideally, 100 times). You should find that roughly half of the rolls are 6s, while the remaining rolls are a made up of the other five values.

Note that when using a sequence of values like in the previous example, we *cannot* enter the values directly into the `choice` function.

```
import random
roll = random.choice(1, 2, 3, 4, 5, 6, 6, 6, 6, 6)
print("You rolled a", roll)
```

This will produce a run-time error.

```
Traceback (most recent call last):
  File "/home/garvin/Code/loaded_die.py", line 2, in <module>
    roll = random.choice(1, 2, 3, 4, 5, 6, 6, 6, 6, 6)
TypeError: choice() takes 2 positional arguments but 11 were given
```

This is because `choice` only accepts a single sequence-type argument. The reason why it was possible to do so using a separate variable is because `loaded_die` is a specific sequence data type known as a **tuple**. We will look at tuples, and other related data types, later in this course.