CSCE 110: Programming I

David Kebo Hougninou

Basics of Python

Randomness
Randomness: Guessing game

Write a program to play the **Guessing Game**, where the goal is to guess a secret number between 1 and 100.

The program generates a secret number between 1 and 100.

If the users guess is wrong the program shows if the guess is too low or too high and keeps asking for a new number.

The program should count and show the number of guesses.

Deterministic vs. non-deterministic programs

Computer programs generate the same outputs every time (deterministic).

To make a program non-deterministic we can use algorithms that generate pseudorandom numbers.

Pseudorandom numbers are not truly random because they are generated by a deterministic computation.
Randomness: random module

Some programs require randomness.

Python comes with a module, called random, that allows us to use random numbers in our programs.

The sequence of number generated depends on a seed.

Because the random number generation is not truly random, the seed is the number that generates the random numbers.

What is a module in Python?

The core part of the Python language consists of:

• if statements
• math operators
• loops
• print
• input

Everything else is contained in modules.

If we want to use something from a module, we import the module.

In other words, we tell Python that we want to use the module.
What is a module in Python?

A module is a file that contains Python code.
e.g.: puzzle.py is a module
The name of the module is puzzle.

To import a module, use the import keyword follows by the name of the module:
import puzzle

Python supports standard and user-defined modules.
Random is a Python standard module.

Randomness: random module

1. Import the module:
   random generates pseudo-random numbers.
To import the functions from the module, use the import keyword
import random

2. Call the functions (use the module name first)
random.randint(1, 100)
random.randint(1, 3)
random.choice(['r', 'p', 's'])
Import operations

We can rename a module
import random as dice
print("Roll it!", dice.randint(1,6))

We can import specific definitions from a module
from random import randint
print("Roll it!", randint(1,6))

We can import all definitions from a module
from random import randint
print("Roll it!", randint(1,6))
print("Guess a letter!", choice('abcdefghij'))

Randomness: random module definitions

The function dir() prints the definitions that are defined inside a module.

List all the definitions in the random module
import random
dir(random)
Randomness: random() function

The function `random()` returns a random float between 0.0 and 1.0 (including 0.0 but not 1.0).

Each time we call `random`, we get the next random number.

```python
>>> for i in range(10):
...     x = random.random()
...     print(x)
0.37490656363282626
0.6634116121253926
0.29025919461346084
0.16700609549374268
0.2700173303486014
0.007769355212417506
0.5057935990564669
0.7082119688523174
0.686471864426977
0.39510459303691936
```

Randomness: randint() function

The function `randint()` takes parameters low and high and returns an integer between low and high (including both).

```python
>>> roll1 = random.randint(5, 10)
>>> roll2 = random.randint(5, 10)
>>> print(roll1)
>>> print(roll2)
6
8
```
Randomness: choice() function

The function `choice()` chooses an element from a sequence at random.

```
>>> t = [1, 2, 3]
... random.choice(t)
3
>>> random.choice(t)
1
```

Examples

```
>>> random.random()  # Random float x, 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(1, 10)  # Random float x, 1.0 <= x < 10.0
1.1800146073117523
>>> random.randint(1, 10)  # Integer from 1 to 10, endpoints included
7
>>> random.randrange(0, 101, 2)  # Even integer from 0 to 100
26
>>> random.choice('abcdefgij')  # Choose a random element 'c'

>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(items)
>>> items
[7, 3, 2, 5, 6, 4, 1]

>>> random.sample([1, 2, 3, 4, 5], 3)  # Choose 3 elements
[4, 1, 5]
```
Problem

The goal of the Guessing Game is to guess a secret number between 1 and 100.

- The program generates a secret number between 1 and 100.
- If the users guess is wrong, the program prompts if the guess is too low or too high and keeps asking for a new number.
- The program should count and show the number of guesses.

Write the pseudocode and the code for this program.
Guessing game: solution

```python
import random

print ('=================================================')
print (' Welcome to The Guessing Game ')
print ('=================================================

secret_num = random.randint (1, 100)
guess = int( input ('Enter a number between 1 and 100: '))

attempts = 1

while (guess != secret_num):
    if (guess < secret_num):
        print ('Too low .'
    else :
        print ('Too high .'
    guess = int( input ('Enter a number between 1 and 100: '))
    attempts += 1

print ('Congratulations ! You got it in',attempts, 'guesses.')
```

Random module: common function

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed(a=None, version=2)</td>
<td>Initialize the random number generator</td>
</tr>
<tr>
<td>getstate()</td>
<td>Returns an object capturing the current internal state of the generator</td>
</tr>
<tr>
<td>setstate(state)</td>
<td>Restores the internal state of the generator</td>
</tr>
<tr>
<td>getrandbits(k)</td>
<td>Returns a Python integer with k random bits</td>
</tr>
<tr>
<td>randrange(start, stop[, step])</td>
<td>Returns a random integer from the range</td>
</tr>
<tr>
<td>randint(a, b)</td>
<td>Returns a random integer between a and b inclusive</td>
</tr>
<tr>
<td>choice(seq)</td>
<td>Return a random element from the non-empty sequence</td>
</tr>
<tr>
<td>shuffle(seq)</td>
<td>Shuffle the sequence</td>
</tr>
<tr>
<td>sample(population, k)</td>
<td>Return a k length list of unique elements chosen from the population sequence</td>
</tr>
<tr>
<td>random()</td>
<td>Return the next random floating point number in the range [0.0, 1.0)</td>
</tr>
<tr>
<td>uniform(a, b)</td>
<td>Return a random floating point number between a and b inclusive</td>
</tr>
</tbody>
</table>