# Uso de Módulos en Python

# Módulos

- *Los Módulos* son archivos que contienen código destinado a ser utilizado en otros programas. Estos módulos generalmente agrupan una colección de programación relacionada con un área. El módulo random contiene funciones relacionadas con la generación de números aleatorios y la producción de resultados aleatorios.
- Cuando importa un módulo, se instala como parte de su instalación de Python o se descarga e instala más tarde, antes de ejecutar el programa que lo importa.

# Importando el módulo random

- La primera línea de código en el programa presenta la declaración import
- La declaración le permite importar o cargar módulos, en este caso, el módulo random:
  - import random

# Generando números aleatorios

- Puedes generar números aleatorios usando :
- randint()
- randrange()

# Función randint()

- Produce un entero aleatorio.
- Observe que el programa no llama directamente a randint ()
- Llama a la función:

```
random.randint(1,6)
```

- Este accede a la función `randint()` a través de su módulo, `random`

# Notación Punto

- Puede llamar a una función desde un módulo importado dando el nombre del módulo, seguido de un punto, seguido de la llamada a la función en sí (similar a los métodos de cadena).
- Este método de acción se llama **notación punto**.
- Al usar la notación punto `random.randint()` significa que la función `randint()` pertenece al módulo random.
- La notación punto se puede usar para acceder a diferentes elementos de módulos importados.

# función randint()

- Requiere dos valores de argumento entero y devuelve un entero aleatorio entre esos dos valores, que puede incluir cualquiera de los valores de argumento.
- Al pasar los valores 1 y 6 a la función randint, tiene la garantía que la máquina retorne 1,2,3,4,5 o 6

# Funcion randrange()

- produce un entero aleatorio
- Usa un solo argumento entero positivo
- La función devuelve un número entero aleatorio desde 0, incluyéndolo hasta ese número, pero sin incluirlo.
- Ejemplo: `random.randrange(6)` produce ya sea 0,1,2,3,4, or 5

# Lanzamiento de dados

- `randrange (6) elige un número aleatorio de un grupo de 6 números, y la lista comienza con 0`
- Si agrega 1 al resultado para obtener el valor correcto para un dado2:
- `dado2=random.randrange(6)+1`

Como resultado, el dado 2 puede obtener uno de estos valores: 1,2,3,4,5 o 6.

# El Módulo `math`

- módulo `math`: Parte de la biblioteca estándar que contiene funciones que son útiles para realizar cálculos matemáticos.
- Por lo general, acepta uno o más valores como argumentos, realiza operaciones matemáticas y devuelve el resultado.
- El uso del módulo requiere una declaración `import math`

# El Módulo `math`

**Table 5-2** Many of the functions in the `math` module

| `math` Module Function | Description |
|---|---|
| `acos(x)` | Returns the arc cosine of `x`, in radians. |
| `asin(x)` | Returns the arc sine of `x`, in radians. |
| `atan(x)` | Returns the arc tangent of `x`, in radians. |
| `ceil(x)` | Returns the smallest integer that is greater than or equal to `x`. |
| `cos(x)` | Returns the cosine of `x` in radians. |
| `degrees(x)` | Assuming `x` is an angle in radians, the function returns the angle converted to degrees. |
| `exp(x)` | Returns $e^x$ |
| `floor(x)` | Returns the largest integer that is less than or equal to `x`. |
| `hypot(x, y)` | Returns the length of a hypotenuse that extends from (0, 0) to (`x`, `y`). |
| `log(x)` | Returns the natural logarithm of `x`. |
| `log10(x)` | Returns the base-10 logarithm of `x`. |
| `radians(x)` | Assuming `x` is an angle in degrees, the function returns the angle converted to radians. |
| `sin(x)` | Returns the sine of `x` in radians. |
| `sqrt(x)` | Returns the square root of `x`. |
| `tan(x)` | Returns the tangent of `x` in radians. |

# El Módulo `math`

- El módulo `math` define variables `pi` y `e`, a los que se les asignan los valores matemáticos para *pi* y *e*
  - Se puede usar en ecuaciones que requieren estos valores, para obtener resultados más precisos
- Las variables también deben llamarse utilizando la notación punto.
  - Ejemplo:

```
Area_circulo = math.pi * radio**2
```

# Asignar Funciones en Módulos

- En programas grandes y complejos, es importante mantener el código organizado.
- Modularización: agrupar funciones relacionadas en módulos.
  - Hace que el programa sea más fácil de entender, probar y mantener.
  - Facilita la reutilización de código para múltiples programas diferentes.
  - Importe el módulo que contiene la función requerida a cada programa que lo necesite.