El futuro digital es de todos

MinTIC

Módulos Random, Fecha y Cadena

Rogerio Orlando Beltrán Castro

Universidad Industrial de Santander

random- Generar números pseudoaleatorios

Este módulo implementa generadores de números pseudoaleatorios para varias distribuciones.

Para los enteros, existe una selección uniforme dentro de un rango. Para las secuencias, existe una selección uniforme de un elemento aleatorio, una función para generar una permutación aleatoria de una lista in-situ y una función para el muestreo aleatorio sin reemplazo.

Para números reales, existen funciones para calcular distribuciones uniformes, normales (Gaussianas), log-normales, exponenciales negativas, gamma y beta. Para generar distribuciones angulares está disponible la distribución de von Mises.

Casi todas las funciones del módulo dependen de la función básica random(), la cuál genera uniformemente un número flotante aleatorio en el rango semiabierto [0.0, 1.0). Python utiliza Mersenne Twister como generador principal. Éste produce números de coma flotante de 53 bits de precisión y tiene un periodo de 2**19937-1. La implementación subyacente en C es rápida y segura para subprocesos. Mersenne Twister es uno de los generadores de números aleatorios más testados que existen. Sin embargo, al ser completamente determinístico, no es adecuado para todos los propósitos y es completamente inadecuado para fines criptográficos.

https://docs.python.org/es/3/library/random.html

Funciones de contabilidad

- random.seed(a=None, version=2)

Inicializa el generador de números aleatorios. Si a es omitida o None, se utilizará la hora actual del sistema. Si las fuentes de aleatoriedad vienen del sistema operativo, éstas se usarán en vez de la hora del sistema (ver la función os.urandom() para detalles sobre su disponibilidad). Si a es un entero, se usará directamente.

- random.getstate()

Retorna un objeto capturando el estado interno del generador. Este objeto puede pasarse a setstate() para restaurar su estado.

- random.setstate(state)

El state debería haberse obtenido de una llamada previa a getstate(), y setstate() reestablece el estado interno del generador al que tenia cuando se llamó a la función getstate().

Funciones para enteros

- random.randrange(stop)
- random.randrange(start, stop[, step])

Retorna un elemento de range(start, stop, step) seleccionado aleatoriamente. Esto es equivalente a choice(range(start, stop, step)), pero en realidad no crea un objeto rango.

El patrón de argumento posicional coincide con el de range(). Los argumentos no deben usarse porque la función puede usarlos de forma inesperada.

- random.randint(a, b)

Retorna un entero aleatorio N tal que a <= N <= b. Alias de randrange(a, b+1).

- random.getrandbits(k)

Returns a non-negative Python integer with k random bits. This method is supplied with the MersenneTwister generator and some other generators may also provide it as an optional part of the API. When available, getrandbits() enables randrange() to handle arbitrarily large ranges.

Distribuciones para los numeros reales

- random.random()

Retorna el siguiente número en coma flotante aleatorio dentro del rango [0.0, 1.0).

- random.uniform(a, b)

Retorna un número en coma flotante aleatorio N tal que a <= N <= b para a <= b y b <= N <= a para b < a. El valor final b puede o no estar incluido en el rango, dependiendo del redondeo de coma flotante en la ecuación a + (b-a) * random().

- random.triangular(low, high, mode)

Retorna un número de coma flotante N tal que low <= N <= high y con el mode especificado entre esos límites. Los límites low (inferior) y high (superior) son por defecto cero y uno. El argumento mode tiene como valor por defecto el punto medio entre los límites, dando lugar a una distribución simétrica.

- random.betavariate(alpha, beta)

Distribución beta. Las condiciones de los parámetros son alpha > 0 y beta > 0. Retorna valores dentro del rango entre 0 y 1.

- random.expovariate(lambd)

Distribución exponencial. lambd es 1.0 dividido entre la media deseada. Debe ser distinto a cero (El parámetro debería llamarse lambda pero esa es una palabra reservada en Python). Retorna valores dentro del rango de 0 a infinito positivo si lambd es positivo, y de infinito negativo a 0 si lambd es negativo.

- random.gammavariate(alpha, beta)

Distribución gamma. (¡No la función gamma!) Las condiciones en los parámetros son alpha > 0 y beta > 0.

statistics — Funciones de estadística matemática

Este módulo proporciona funciones para calcular estadísticas matemáticas de datos numéricos (de tipo Real).

Este módulo no pretende ser competidor o sustituto de bibliotecas de terceros como NumPy o SciPy, ni de paquetes completos de software propietario para profesionales como Minitab, SAS o Matlab. Este módulo se ubica a nivel de calculadoras científicas gráficas.

A menos que se indique explícitamente lo contrario, las funciones de este módulo manejan objetos int, float, Decimal y Fraction.

https://docs.python.org/es/3/library/statistics.html

Promedios y medidas de tendencia central

Estas funciones calculan el promedio o el valor típico de una población o muestra.

`mean()`	Media aritmética («promedio») de los datos.
`fmean()`	Media aritmética usando coma flotante, más rápida.
`geometric_mean()`	Media geométrica de los datos.
`harmonic_mean()`	Media armónica de los datos.
`median()`	Mediana (valor central) de los datos.
`median_low()`	Mediana baja de los datos.
`median_high()`	Mediana alta de los datos.
`median_grouped()`	Mediana, o percentil 50, de los datos agrupados.
`mode()`	Moda única (valor más común) de datos discretos o nominales.
`multimode()`	Lista de modas (valores más comunes) de datos discretos o nominales.
`quantiles()`	Divide los datos en intervalos equiprobables.

datetime — Tipos básicos de fecha y hora

El módulo datetime proporciona clases para manipular fechas y horas.

Si bien la implementación permite operaciones aritméticas con fechas y horas, su principal objetivo es poder extraer campos de forma eficiente para su posterior manipulación o formateo.

función datetime.timedelta ()

La timedelta()función de Python está presente en la biblioteca de fecha y hora, que generalmente se usa para calcular diferencias en las fechas y también se puede usar para manipulaciones de fechas en Python. Es una de las formas más sencillas de realizar manipulaciones de fechas.

datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)

Ejemplo función datetime.timedelta ()

```
#Demostración de la función Timedelta
from datetime import datetime, timedelta
# Utilizar la fecha actual
hoy = datetime.now()
# Imprime el dia de hoy
print ("Fecha inicial", str(hoy))
# Calculado fecha futuras
# para dos años
fecha_futura_2_annios = hoy + timedelta(days = 730)
fecha_futura_2_dias = hoy + timedelta(days = 2)
# printing calculated future_dates
print('Fecha a dos años:', str(fecha_futura_2_annios))
print('FEcha a dos dias:', str(fecha_futura_2_dias))
```

OBTENER LA FECHA ACTUAL.

```
#Es necesario importar las depencendias necesarias
from datetime import date
from datetime import datetime
#Día actual
today = date.today()
#Fecha actual
now = datetime.now()
print(today)
print(now)
```

ATRIBUTOS: Una vez obtengamos la fecha actual podremos obtener el día, mes, año, hora, minutos, segundos de esta.

```
#Es necesario importar las depencendias necesarias
from datetime import date
from datetime import datetime
#Día actual
today = date.today()
#Fecha actual
now = datetime.now()
#Date
print("El día actual es {}".format(today.day))
print("El mes actual es {}".format(today.month))
print("El año actual es {}".format(today.year))
#Datetime
print("El día actual es {}".format(now.day))
print("El mes actual es {}".format(now.month))
print("El año actual es {}".format(now.year))
print("La hora actual es {}".format(now.hour))
print("El minuto actual es {}".format(now.minute))
print("El segundo actual es {}".format(now.second))
```

NUEVA FECHA: Si nosotros así lo deseamos podemos trabajar con una fecha en particular.

```
#Es necesario importar las depencias necesarias
import datetime


nueva_fecha = datetime.datetime(2019, 2, 28, 10, 15, 0, 00000)
print(nueva_fecha)
# Los argumentos
# serán: Año, Mes, Día, Hora, Minutos, Segundos, Milisegundos.
```

OPERACIONES

En ocasiones tendremos la necesidad de realizar ciertas operaciones con fechas, ya sea agregar días, restar años o simplemente realizar comparaciones. Con Python todas estas operaciones podremos realizarlas sin ningún problema.

```
from datetime import datetime
from datetime import timedelta
#Sumar dos días a la fecha actual
now = datetime.now()
new_date = now + timedelta(days=2)
print(new_date)
#Comparación
if now < new_date:
    print("La fecha actual es menor que la nueva fecha")
```

FORMATO DE FECHAS

Aunque las fechas en Python ya poseen un formato legible para los humanos, en ocasiones necesitaremos ser más explícitos para no dejar espacio a la ambigüedad, para ello, haremos uso del método strftme.

%d Día %m Mes %Y Año

%H Hora %M Minutos %S segundos

```
from datetime import datetime, timedelta

now = datetime.now()

format = now.strftime('Día :%d, Mes: %m, Año: %Y, Hora: %H, Minutos: %M, Segundos: %S')

print(format)
```

Mostrar fecha y hora (datetime)

```
from datetime import datetime
ahora = datetime.now()  # Obtiene fecha y hora actual
print("Fecha y Hora:", ahora)  # Muestra fecha y hora
print("Fecha y Hora UTC:",ahora.utcnow())  # Muestra fecha/hora UTC
print("Día:",ahora.day)  # Muestra día
print("Mes:",ahora.month)  # Muestra mes
print("Año:",ahora.year)  # Muestra año
print("Hora:", ahora.hour)  # Muestra hora
print("Minutos:",ahora.minute)  # Muestra minuto
print("Segundos:", ahora.second)  # Muestra segundo
print("Microsegundos:",ahora.microsecond)  # Muestra microsegundo
```

Comparando fechas y horas (datetime, date)

```
from datetime import datetime, date, time, timedelta
print("Horas:")
hora1 = time(10, 5, 0)  # Asigna 10h 5m 0s
print("\tHora1:", hora1)
hora2 = time(23, 15, 0)  # Asigna 23h 15m 0s
print("\tHora2:", hora2)
# Compara horas
print("\tHora1 < Hora2:", hora1 < hora2)  # True
print("Fechas:")
fecha1 = date.today()  # Asigna fecha actual
print("\tFecha1:", fecha1)
# Suma a la fecha actual 2 días
fecha2 = date.today() + timedelta(days=2)
print("\tFecha2:", fecha2)
# Compara fechas
print("\tFecha1 > Fecha2:", fecha1 > fecha2)  # False
```

cadena al objeto de fecha y hora

```
from datetime import datetime

cadena_fecha = "21/06/2018"

print("Fecha en cadena =", cadena_fecha)
print("Tipo de fecha cadena =", type(cadena_fecha))

fecha = datetime.strptime(cadena_fecha, "%d/%m/%Y")

print("Fecha =", fecha)
print("Tipo =", type(fecha))
```

Otros ejemplos de operaciones con otras unidades de tiempo

```
from datetime import datetime, date, time, timedelta
hoy = date.today()  # Asigna fecha actual
ayer = hoy-timedelta(days=1)  # Resta a fecha actual 1 día
mañana = hoy + timedelta(days=1)  # Suma a fecha actual 1 día
diferencia_en_dias = mañana-hoy  # Resta las dos fechas
hoy_mas_1_millon_segundos = hoy + timedelta(seconds=1000000)
ahora = datetime.now()
hora_actual = time(ahora.hour, ahora.minute, ahora.second)
mas_5m = ahora + timedelta(seconds=300)
mas_5m = time(mas_5m.hour, mas_5m.minute, mas_5m.second)
racion_de_5h = timedelta(hours=5)
mas_5h = ahora + racion_de_5h
print("Ayer:", ayer)
print("Hoy:", hoy)
print("Mañana:", mañana)
print("Diferencia en días entre mañana y hoy:",diferencia_en_dias.days)
print("La fecha de hoy más 1 millón de segundos:",hoy_mas_1_millon_segundos)
print("Hora actual:", hora_actual)
print("Hora actual + 5 minutos:", mas_5m)
print("Hora actual + 5 horas:", mas_5h)
```

Diferencia de dos fechas en días, introducidas por teclado

```
from datetime import datetime
formato = "%d/%m/%Y" # Establecer formato de las fechas a introducir: dd/mm/aaaa
while True: # Bucle 'sin fin'
    try:
        fecha_desde = input('Introducir fecha inicial (dd/mm/aaaa): ')# Introducir fecha inicial utilizando el formato definido
        if fecha_desde == "": # Si no se introduce ningún valor se fuerza el final del bucle
            break
        fecha_hasta = input('Introducir fecha final   (dd/mm/aaaa): ')# Introducir fecha final utilizando el formato definido
        if fecha_hasta == "":# Si no se introduce ningún valor se fuerza el final del bucle
            break
# Se evaluan las fechas según el formato dd/mm/aaaa
# En caso de introducirse fechas incorrectas se capturará
# la excepción o error
        fecha_desde = datetime.strptime(fecha_desde, formato)
        fecha_hasta = datetime.strptime(fecha_hasta, formato)
        if fecha_hasta >= fecha_desde:# Se comprueba que fecha_hasta sea mayor o igual que fecha_desde
            diferencia = fecha_hasta - fecha_desde # Se cálcula diferencia en día y se muestra el resultado
            print("Diferencia:", diferencia.days, "días")
        else:
            print("La fecha fecha final debe ser mayor o igual que la inicial")
    except:
        print('Error en la/s fecha/s. ¡Inténtalo de nuevo!')
```

Diferencia de dos fechas (date)

```
from datetime import datetime,date
hoy = date.today()
navidad_año_proximo = date(2022, 12, 25)
faltan = navidad_año_proximo - hoy
print ("Hoy:", hoy)
print ("La navidad del 2024", navidad_año_proximo)
print ("Faltan", faltan.days, "días")
```

Obtener día de la semana por su número

La función weekday() devuelve el número de día de la semana a que corresponda la fecha indicada, según los siguientes valores por día: 0-Lunes, 1-Martes, 2-Miércoles, 3-Jueves, 4-Viernes , 5-Sábado y 6-Domingo

Obtener y contar los días que sean martes entre dos fechas

```
from datetime import datetime, timedelta
formato = "%d/%m/%Y"
contador = 0
fechadesde = input('Fecha desde (dd/mm/aaaa): ')
fechahasta = input('Fecha hasta (dd/mm/aaaa): ')
if fechadesde == '' or fechahasta == '':
    exit()
try:
    fechadesde = datetime.strptime(fechadesde, formato)
    fechahasta = datetime.strptime(fechahasta, formato)
    if fechadesde > fechahasta:
        print('Fecha desde debe ser menor o igual que hasta')
while fechadesde <= fechahasta:
        if datetime.weekday(fechadesde) == 1:
            contador +=1
            fechaactual = fechadesde.strftime(formato)
            print(contador, fechaactual, 'es martes')
        fechadesde = fechadesde + timedelta(days=1)
except:
    print('Fecha incorrecta')
```

Funciones Incluidas

		Funciones Built-in		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Métodos de las cadenas de caracteres

Todas las cadenas de caracteres implementan las operaciones comunes de las secuencias, junto con los métodos descritos a continuación.

Las cadenas soportan dos estilos de formateo, uno proporciona un grado muy completo de flexibilidad y personalización (Véase str.format(), Sintaxis de formateo de cadena y Formato de cadena de caracteres personalizado) mientras que el otro se basa en la función C printf, que soporta un menor número de tipos y es ligeramente más complicada de usar, pero es a menudo más rápida para los casos que puede manejar (Formateo de cadenas al estilo *printf*).

Métodos de las cadenas de caracteres

- str.capitalize()

Retorna una copia de la cadena con el primer carácter en mayúsculas y el resto en minúsculas.

- str.casefold()

Retorna el texto de la cadena, normalizado a minúsculas. Los textos así normalizados pueden usarse para realizar búsquedas textuales independientes de mayúsculas y minúsculas.

- str.center(width[, fillchar])

Retorna el texto de la cadena, centrado en una cadena de longitud width. El relleno a izquierda y derecha se realiza usando el carácter definido por el parámetro fillchar (Por defecto se usa el carácter espacio ASCII). Si la cadena original tiene una longitud len(s) igual o superior a width, se retorna el texto sin modificar.

- str.count(sub[, start[, end]])

Retorna el número de ocurrencias no solapadas de la cadena sub en el rango [start, end]. Los parámetros opcionales start y end Se interpretan como en una expresión de rebanada.

Métodos de las cadenas de caracteres

- str.find(sub[, start[, end]])

Retorna el menor índice de la cadena s donde se puede encontrar la cadena sub, considerando solo el intervalo s[start:end]. Los parámetros opcionales start y end se interpretan como si fueran “indices de una rebanada. retorna -1 si no se encuentra la cadena..

- str.replace(old, new[, count])

Retorna una copia de la cadena con todas las ocurrencias de la cadena old sustituidas por new. Si se utiliza el parámetro count, solo se cambian las primeras count ocurrencias.

Ejemplos Métodos de las cadenas de caracteres

- frase=input('Frase:')

```
print(frase.capitalize())
print(frase.casefold())
print(frase.center(100,'0'))
print(frase.count('a',1,100))
print(frase.find('a',1,100))
print(frase.replace('a','A',5))
```