Modules

🎯 Session Objectives:

- ✅ Understand what a Python module is
- ✅ Create and use your own module
- ✅ Import specific functions or all functions from a module
- ✅ Rename a module or function using an alias
- ✅ Understand and use built-in modules

1 Module

A module is just a single Python file (.py) that contains code – functions, classes, or variables – that you can import into another Python file.

Purpose: Organize code into reusable pieces.

2 Package

A package is a folder containing multiple Python modules and a special `__init__.py` file (even if empty).

It helps organize modules into a hierarchical structure.

3 Library

A library is a collection of related modules and packages that provide a set of functionalities.

Example: NumPy, Pandas

4 Framework

A framework is a bigger structure that provides not only modules and packages but also a set of rules and architecture for building applications.

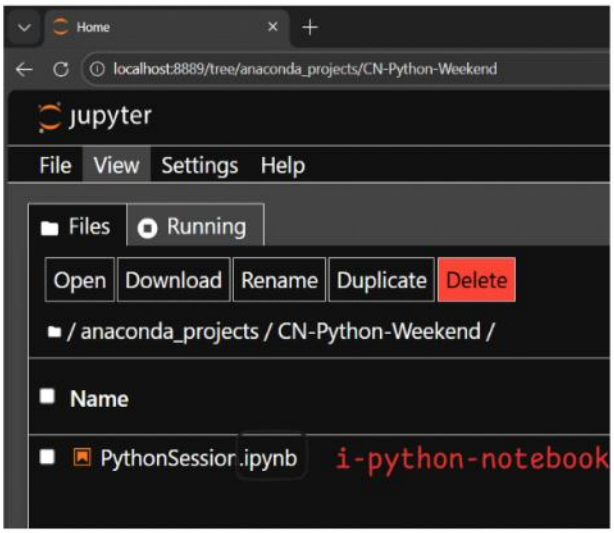
Example: Django, Flask (web frameworks).

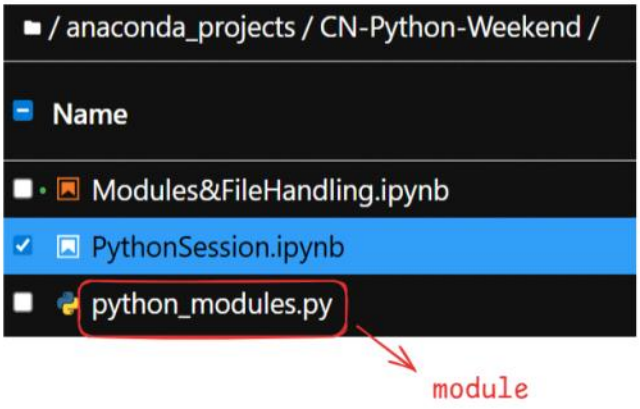
What is a Module?

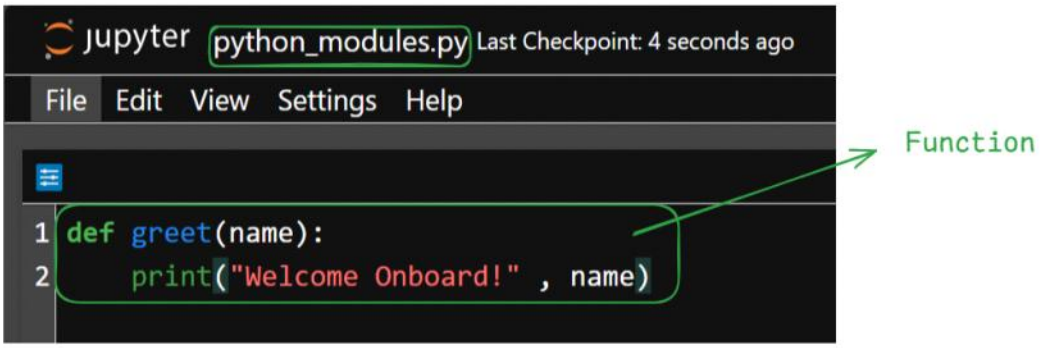
A module is just a .py file that contains reusable Python code—like functions, variables, or classes.

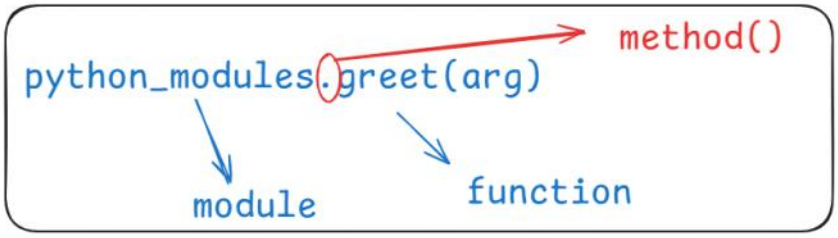
Why Use Modules?

- Avoid writing the same code again and again.
- Break large programs into smaller, manageable pieces.
- Share functionality across different files
- Make code cleaner, organized, and reusable









What is a Module? ¶

A module is just a .py file that contains reusable Python code—like functions, variables, or classes.

Why Use Modules?

- Avoid writing the same code again and again.
- Break large programs into smaller, manageable pieces.
- Share functionality across different files
- Make code cleaner, organized, and reusable

Analogy:

- `Module` - A single Chapter in a book.
- `Package` - A book made of multiple Chapters.
- `Library` - A bookshelf containing multiple books on a subject.
- `Framework` - A Whole Library Building with a set of rules of how books are organized and used.

```
import python_modules
python_modules.greet('Shyam Sundar')
```

```
Welcome Onboard! Shyam Sundar
```

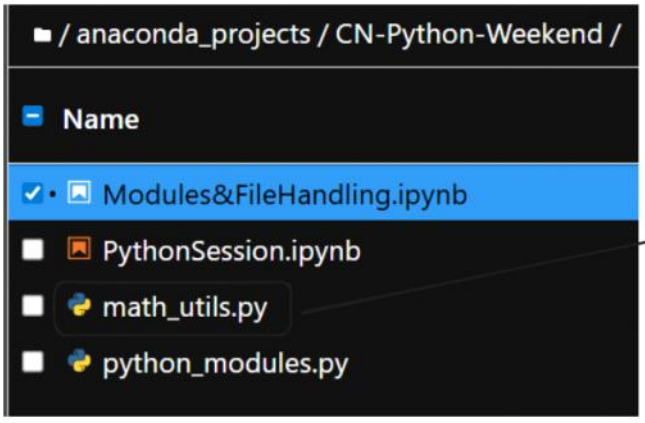
```
python_modules.greet('Rajat Singh')
```

```
Welcome Onboard! Rajat Singh
```

```
python_modules.greet('Ankita Dutta')
```

```
Welcome Onboard! Ankita Dutta
```

User Defined



A Single Chapter in a book
[Topics -> Functions]

jupyter math_utils.py Last Checkpoint: 11 minutes ago

File Edit View Settings Help

```python
def add(x,y):
    return x + y

def sub(x,y):
    return x - y

def multiply(x,y):
    return x * y

def divide(x,y):
    return x // y

def square(val):
    return val ** 2

def cube(val):
    return val ** 3
```

```python
def fact(n):
    if n < 0:
        raise ValueError("n in factorial can't be negative.")
    elif n == 1:
        return 1
    else:
        return n * fact(n-1) # Recursive Call
```

```python
import math_utils # Importing .py file [Module]
result = math_utils.add(10,11)
print(result)
```

```
21
```

```python
result = math_utils.sub(21,11)
print(result)
```

```
10
```

```python
result = math_utils.multiply(10,11)
print(result)
```

```
110
```

```python
result = math_utils.divide(110,11)
print(result)
```

```
10
```

```python
result = math_utils.square(11)
print(result)
```

```
121
```

```python
result = math_utils.cube(7)
print(result)
```

```
343
```

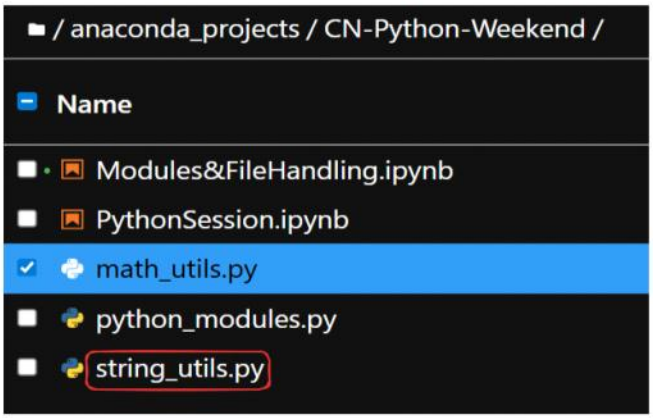
```
result = math_utils.fact(-7)
print(result)
```

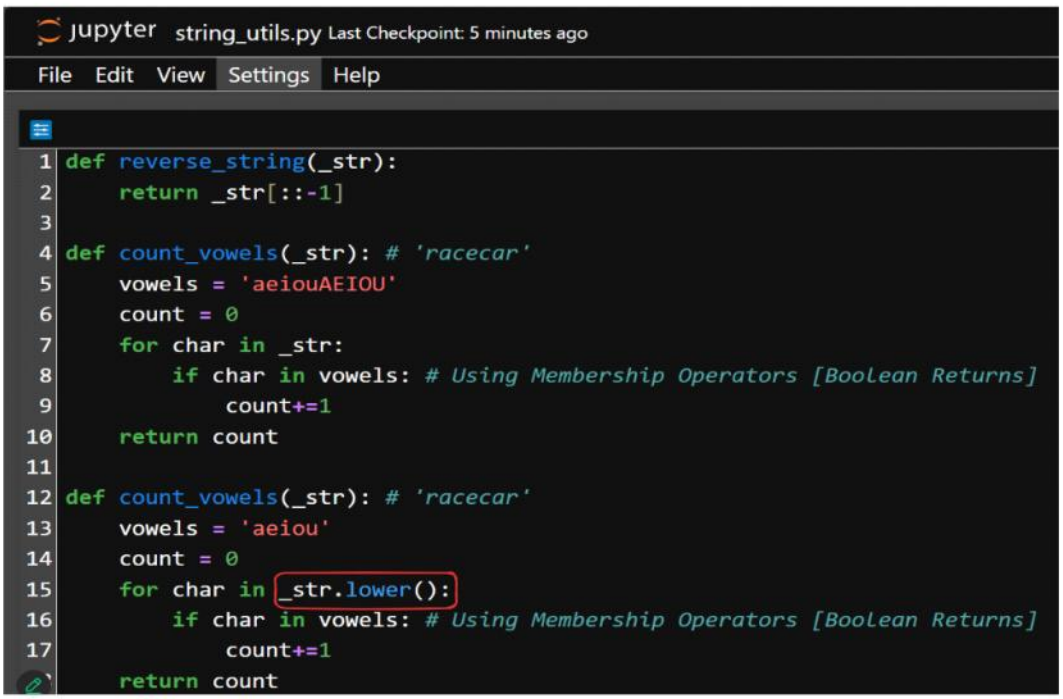
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 1
----> 1 result = math_utils.fact(-7)
      2 print(result)

File ~\anaconda_projects\CN-Python-Weekend\math_utils.py:21, in fact(n)
     19 def fact(n):
     20     if n < 0:
---> 21         raise ValueError("n in factorial can't be negative.")
     22     elif n == 1:
     23         return 1

ValueError: n in factorial can't be negative.
```

Fix Code

```
result = math_utils.fact(5)
print(result)
```

```
120
```






```python
def reverse_string(_str):
    return _str[::-1]

def count_vowels(_str): # 'racecar'
    vowels = 'aeiouAEIOU'
    count = 0
    for char in _str:
        if char in vowels: # Using Membership Operators [Boolean Returns]
            count+=1
    return count

def count_vowels(_str): # 'racecar'
    vowels = 'aeiou'
    count = 0
    for char in _str.lower():
        if char in vowels: # Using Membership Operators [Boolean Returns]
            count+=1
    return count
```

```
import string_utils
rev_str = string_utils.reverse_string('racecar')
print(rev_str)
```

```
racecar
```

```
rev_str = string_utils.reverse_string('coding ninjas')
print(rev_str)
```

```
sajnin gnidoc
```

```
rev_str = string_utils.reverse_string('hello')
print(rev_str)
```

```
olleh
```

```
count_vowels = string_utils.count_vowels('Coding Ninjas') # 4
print(count_vowels)
```

```
4
```

```
count_vowels = string_utils.count_vowels('Python Programming') # 4
print(count_vowels)
```

```
4
```

```
count_vowels = string_utils.count_vowels('Today is a very awesome day!') # 10
print(count_vowels)
```

```
10
```

```
# Calling the function from a module directly
from string_utils import reverse_string , count_vowels
rev_str = reverse_string('Python')
print(rev_str)
```

```
nohtyP
```

```
# Calling the function from a module directly
from string_utils import reverse_string , count_vowels
rev_str = reverse_string('Python')
print(rev_str)
```

```
nohtyP
```

```
count_vowels = count_vowels('Shyam Sundar Rao Majji') # 7
print(count_vowels)
```

```
7
```

```
# SELECT * FROM TABLE; -> '*' - all columns
from math_utils import *
_sum = add(20,57) # 77
print(_sum)
```

```
77
```

```
_mul = multiply(10,5)
_mul
```

```
50
```

```
# Aliases -> 'as' keyword
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import math_utils as math
math.add(10,11)
```

```
21
```

```
math.square(10)
```

```
100
```

```
math.cube(11)
```

```
1331
```

What Are Built-in Modules?

Python includes standard modules that come pre-installed.
You don't have to download them—you just import and use them.

They help with:

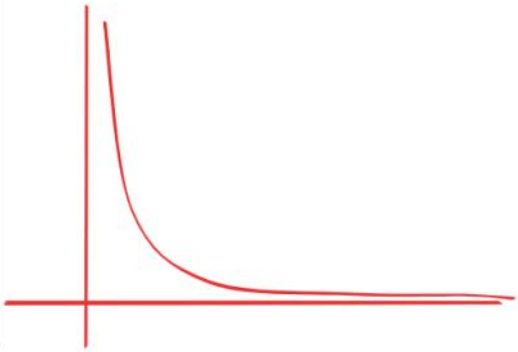
- Getting system or platform information
- Performing mathematical operations
- Handling dates and times
- Generating random values
- Working with files and directories

Trigonometry Table

Angles (in Degrees)	0°	30°	45°	60°	90°	180°	270°
Angles (in Radians)	0	π/6	π/4	π/3	π/2	π	3π/2
sin	0	1/2	1/√2	√3/2	~~-1~~ 1	0	-1
cos	1	√3/2	1/√2	1/2	0	-1	0
tan	0	1/√3	1	√3	Not Defined	0	Not Defined
cot	Not Defined	√3	1	1/√3	0	Not Defined	0
cosec	Not Defined	2	√2	2/√3	-1	Not Defined	-1
sec	1	2/√3	√2	2	Not Defined	-1	Not Defined

Trigonometry Table

	0°	30°	45°	60°	90°
sin θ	0	$\frac{1}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{\sqrt{3}}{2}$	1
cos θ	1	$\frac{\sqrt{3}}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	0
tan θ	0	$\frac{1}{\sqrt{3}}$	1	$\sqrt{3}$	Not defined
cosec θ	Not defined	2	$\sqrt{2}$	$\frac{2}{\sqrt{3}}$	1
sec θ	1	$\frac{2}{\sqrt{3}}$	$\sqrt{2}$	2	Not defined
cot θ	Not defined	$\sqrt{3}$	1	$\frac{1}{\sqrt{3}}$	0



```
0.999999
math.ceil
1
```

```
Docstring:
log(x, [base=math.e])
Return the logarithm of x to the given base.

If the base is not specified, returns the natural logarithm (base e) of x.
Type:      builtin_function_or_method
```

```
math.log(100)
```

$$\log_{10}(100000)$$

$$\log_{10}(10^5) = 5(\log_{10}10) \quad (\log_{10}10 = 1)$$

What Are Built-in Modules?

Python includes standard modules that come pre-installed. You don't have to download them—you just import and use them.

They help with:

- Getting system or platform information
- Performing mathematical operations
- Handling dates and times
- Generating random values
- Working with files and directories

```
# Getting system or platform information
import platform
dir(platform) # 'gist' of what functions stores in this 'platform' module
```

```
 'architecture',
 'collections',
 'freedesktop_os_release',
 'functools',
 'itertools',
 'java_ver',
 'libc_ver',
 'mac_ver',
 'machine',
 'node',
 'os',
 'platform',
 'processor',
 'python_branch',
 'python_build',
 'python_compiler',
 'python_implementation',
 'python_revision'
```

```
platform.__name__
```

```
'platform'
```

```
platform.__version__
```

```
'1.0.8'
```

```
platform.__doc__
```

```
' This module tries to retrieve as much platform-identifying data as\n    possible. It makes this information
available via function APIs.\n\n    If called from the command line, it prints the platform\n    information c
oncatenated as single string to stdout. The output\n    format is usable as part of a filename.\n\n'
```

```
import platform as p
p.system()
```

```
'Windows'
```

```
p.version()
```

```
'10.0.26200'
```

```
p.release()
```

```
'11'
```

```
p.machine()
```

```
'AMD64'
```

```
p.architecture()
```

```
('64bit', 'WindowsPE')
```

```
p.platform()
```

```
'Windows-11-10.0.26200-SP0'
```

```
p.processor()
```

```
'Intel64 Family 6 Model 186 Stepping 3, GenuineIntel'
```

```
p.uname()
```

```
uname_result(system='Windows', node='Priya', release='11', version='10.0.26200', machine='AMD64')
```

```
p.python_version()
```

```
'3.12.7'
```

```
p.python_compiler()
```

```
'MSC v.1929 64 bit (AMD64)'
```

```
p.python_implementation() # C Language
```

```
'CPython'
```

```
p.mac_ver()
```

```
('', ('', '', ''), '')
```

```
# Performing mathematical operations
import math
dir(math)
```

```
 'nan',
 'nextafter',
 'perm',
 'pi',
 'pow',
 'prod',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'sumprod',
 'tan',
 'tanh',
 'tau',
 'trunc',
 'ulp']
```

```
math.ceil(199.79) # 200
```

```
200
```

```
math.floor(199.81) # 199
```

```
199
```

```
math.fabs(-199.99) # |-1| = 1 # Return float with absolute
```

```
199.99
```

```
math.factorial(5)
```

```
120
```

```
math.factorial(7)
```

```
5040
```

```
math.sqrt(81) # 9
```

```
9.0
```

```
math.sqrt(121) # 11
```

```
11.0
```

```
math.gcd(12,18) # 6
```

```
6
```

```
math.gcd(15,25) # 5
```

```
5
```

```
math.lcm(12,18) # 2*2*3*3 = 36
```

```
36
```

```
math.pi
```

```
3.141592653589793
```

```
math.radians(180)
```

```
3.141592653589793
```

```
# Parimeter of a circle -> 2 * pi * r => radius [1cm] -> 2 * pi -> 360 degree
math.radians(360)
```

```
6.283185307179586
```

```
math.sin(math.pi/2) # sin90 - 1
```

```
1.0
```

```
math.tan(math.pi/4) # tan45 - 1
```

```
0.9999999999999999
```

```
math.ceil(math.tan(math.pi/4)) # Round Up
```

```
1
```

```
math.log(100,10) # 10^2 = 2k
```

```
2.0
```

```
math.log(100000,10) # 10^5 = 5
```

```
5.0
```

```
math.log10(1000) # 3
```

```
3.0
```

```
# Handling dates and times
import datetime
dir(datetime)
```

```
['MAXYEAR',
 'MINYEAR',
 'UTC',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'date',
 'datetime',
 'datetime_CAPI',
 'time',
 'timedelta',
 'timezone',
 'tzinfo']
```

```
datetime.MINYEAR
```

```
1
```

```
datetime.MAXYEAR
```

```
9999
```

```
dir(datetime.date)
```

```
 'fromisoformat',
 'fromordinal',
 'fromtimestamp',
 'isocalendar',
 'isoformat',
 'isoweekday',
 'max',
 'min',
 'month',
 'replace',
 'resolution',
 'strftime',
 'timetuple',
 'today',
 'toordinal',
 'weekday',
 'year']
```

```
dir(datetime.datetime)
```

```
 'microsecond',
 'min',
 'minute',
 'month',
 'now',
 'replace',
 'resolution',
 'second',
 'strftime',
 'strptime',
 'time',
 'timestamp',
 'timetuple',
 'timetz',
 'today',
 'toordinal',
 'tzinfo',
```

```
dir(datetime.timedelta)
```

```
 '__rmod__',
 '__rmul__',
 '__rsub__',
 '__rtruediv__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 'days',
 'max',
 'microseconds',
 'min',
 'resolution',
 'seconds',
 'total_seconds']
```

```
now = datetime.datetime.now()
print(now)
```

```
2026-01-17 13:26:17.414556
```

```
now
```

```
datetime.datetime(2026, 1, 17, 13, 26, 17, 414556)
```

```
today =  datetime.datetime.today()
today
```

```
datetime.datetime(2026, 1, 17, 13, 28, 5, 348822)
```

```
print(today)
```

```
2026-01-17 13:28:05.348822
```

```
# strftime [formatting]
# 2026-01-17 13:26
formatted = datetime.datetime.now().strftime("%Y-%m-%d %H:%M")
print(formatted)
```

```
2026-01-17 13:32
```

```
formatted = now.strftime("%Y-%m-%d %H:%M")
print(formatted)
```

```
2026-01-17 13:26
```

```
# "DayName, 2026-01-17"
formatted = now.strftime("%A, %Y-%m-%d")
print(formatted)
```

```
Saturday, 2026-01-17
```

```
# strptime
formatted = datetime.datetime.strptime("2026-01-17 13:28:05" , "%Y-%m-%d %H:%M:%S")
print(formatted)
```

```
2026-01-17 13:28:05
```

```
# Dateadd() or Datediff()
from datetime import timedelta
tomorrow = today +  timedelta(days = 1)
print(tomorrow)
```

```
2026-01-18 13:28:05.348822
```

```
yesterday = today -  timedelta(days = 1)
print(yesterday)
```

```
2026-01-16 13:28:05.348822
```

```
# 2026-01-17 13:26:17
change_in_seconds = now +  timedelta(seconds = 600) # 10 minute
print(change_in_seconds)
```

```
2026-01-17 13:36:17.414556
```

```
# Generating random values
import random
dir(random)
```

```
  'expovariate',
  'gammavariate',
  'gauss',
  'getrandbits',
  'getstate',
  'lognormvariate',
  'normalvariate',
  'paretovariate',
  'randbytes',
  'randint',
  'random',
  'randrange',
  'sample',
  'seed',
  'setstate',
  'shuffle',
  'triangular',
  'uniform',
```

```
random.random() # float value between (0,1) exclusive
```

```
0.32923533617242307
```

```
random.random()
```

```
0.08923151727829237
```

```
random.random()
```

```
0.044422086506070046
```

```
random.random()
```

```
0.6044594125094763
```

```
random.randint(1,6) # Both Inclusive
```

```
2
```

```
random.randint(1,6) # Both Inclusive
```

```
4
```

```
random.randint(1,6) # Both Inclusive
```

```
1
```

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

```
6
```

```
random.randint(1,500) # Range Increased [1,500]
```

```
439
```

```
random.uniform(0.11 , 9.99) # float value between provided range
```

```
9.26672693873923
```

```
random.uniform(0.11 , 9.99)
```

```
2.2275703940290694
```

```
random.uniform(0.11 , 9.99)
```

```
2.091899026798806
```

```
# choice # k-factor = 1
car_list = ['ScorpioN', 'Creta','Hummer','Defender','Thar','Seltos','Elevate','Sierra',
            'Alto','Taigun','Slavia','Verna','Virtus','3X0','Bolero','Fronx','Magnite','Jimny']
random.choice(car_list)
```

```
'Slavia'
```

```
random.choice(car_list)
```

```
'Jimny'
```

```
random.choice(car_list)
```

```
'Virtus'
```

```
random.choice(car_list)
```

```
'Bolero'
```

```
# choices [k-factor] [default = 1]
random.choices(car_list , k=7) # 7 elements in list [repeated]
```

```
['3X0', 'Verna', 'Magnite', 'Hummer', 'Verna', 'Bolero', 'Defender']
```

```
random.choices(car_list , k=5)
```

```
['Alto', 'Slavia', 'Defender', 'Fronx', 'Slavia']
```

```
random.choices(car_list , k=7)
```

```
['ScorpioN', 'Taigun', 'ScorpioN', 'Jimny', 'Verna', 'Bolero', 'Creta']
```

```
random.choices(car_list , k=7)
```

```
['Creta', 'Magnite', '3X0', 'Hummer', 'Magnite', 'Verna', 'Verna']
```

```
random.choices(car_list , k=7)
```

```
['Slavia', 'Seltos', 'ScorpioN', 'Virtus', 'Alto', 'Verna', 'Defender']
```

```
random.choices(car_list , k=7)
```

```
['Elevate', 'Taigun', 'Fronx', 'Elevate', 'Sierra', 'Taigun', 'Creta']
```

```
# shuffle
_cards = ['Ace','1','2','3','4','5','6','7','8','9','10','Jack','Queen','King']
random.shuffle(_cards)
print(_cards)
```

```
['4', '10', '9', 'Ace', '7', '3', 'Jack', '8', '2', 'Queen', '1', '6', '5', 'King']
```

```
random.shuffle(_cards)
print(_cards)
```

```
['1', 'Ace', '7', '3', 'Queen', '4', 'King', '2', '9', '5', '10', '8', 'Jack', '6']
```

```
random.shuffle(_cards)
print(_cards)
```

```
['7', 'King', '5', '3', '2', 'Queen', '1', '10', '4', '8', '9', 'Ace', '6', 'Jack']
```

```
# seed -> Setting seed for reproducibility:
random.seed(42)
random.random()
```

```
0.6394267984578837
```

```
random.seed(41)
random.random()
```

```
0.38102068999577143
```

```
random.seed(511)
random.random()
```

```
0.10846537689444802
```