The pass Statement

- serves as a placeholder for future code, preventing errors from empty code blocks.
- be typically used where code is planned but has yet to be written.

```
def future_function():
  pass

future_function()
```

Function With Arbitrary (*) Arguments

- do not know the number of arguments that will be passed into a function. ---> use arbitrary arguments in
- allow to pass a varying number of values during a function call.
- use an asterisk (*) before the parameter name

```
def find_sum(*numbers):
  result = 0

  for num in numbers: result += num
  print(f"Sum: {result}")

find_sum(1, 2, 3)
find_sum(2, 4, 5, 10)
```

```
Sum: 6
Sum: 21
```

Local Variables

- inside a function, local scope (within the function)
- cannot access outside the function

Global Variables

- outside of the function or in global scope
- can be accessed inside or outside of the function.

Nonlocal Variables

- The nonlocal keyword is used within nested functions to indicate that a variable is not local to the inner function, but rather belongs to an enclosing function's scope.
- This allows you to modify a variable from the outer function within the nested function, while still keeping it distinct from global variables.

Nest function

- a function defined inside the body of another function

```
#global
newmess = "Hi"
```

```
def greet():
  #local variable
  message = "Hello"

  #nest function
  def water():
    #declare nonlocal variable
    nonlocal message
    message = "Give me a cup of water"

  print(f"inner {message}")
  water()
  print(f"outer {message}")

greet()
```

```
inner Hello
outer Give me a cup of water
```

Recursive Function

- the process of defining something in terms of itself, a function can call other functions.
- It is even possible for the function to call itself.

Advantages of Recursion

- make the code look clean and elegant
- A complex task can be broken down into simpler sub-problems
- Sequence generation is easier with recursion than using nested iteration.

Disadvantages of Recursion

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- hard to debug.

```
# factorial
def factorial(x):
  if x == 1:
    return 1
  else:
    return (x*factorial(x-1))

num = 5
print(f"The factorial of {num} is {factorial(num)}")
```

```
The factorial of 5 is 120
```

Package

- Packages ---> Modules ---> Classes ---> Functions. e.g. Math pckage includes sqrt()
- Big projects: large amount of code, if do everything in just 1 file ---> messy ---> seperate into multiple files (keeping the related code in pakages)

```
package{game}
  __init__.py
  subpackage{Sound}
    __init__.py
    load.py ---> module
    play.py ---> module
  subpakage{}
  ...

def add(a, b) ---> function
  return a + b
```

Import module

```
import Game.Level.Start
```

- similar to

```
from Game.Level import Start
```

- import function only

```
from Game.Level.Start import select_difficulty
select_difficulty(2)
```

- import a module

```
import math
print(math.pi)

#or

import math as m
print(m.pi)
```

- Import all function in a module

```
from math import *
print(pi)
```

List all functions in a module

- e.g. list all function in "math" module

```
print(dir(math))
```

Name of the module

```python
import matplotlib.pyplot as plt
print(plt.__name__)

import math
import matplotlib.pyplot as plt
print(dir(math))
print(plt.__name__)
print(plt)
print(__name__)
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt',
'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e',
'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm',
'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt',
'sumprod', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
matplotlib.pyplot
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.12/dist-
packages/matplotlib/pyplot.py'>
__main__
```

Main function

- Define the execution point

```python
if __name__ == '__main__':
  # code
  # main()
```

Run python file from the command line

```
$ python helloworld.py
```

Running Python file as a Module

```python
import math
math
```

NumPy

- provides a basic array data type (think of vectors and matrices) and functions for acting on these arrays (e.g., matrix multiplication).

SciPy

- builds on NumPy by adding numerical methods routinely used in science (interpolation, optimization, root finding, etc.).

Matplotlib

- generate figures, with a focus on plotting data stored in NumPy arrays

JAX

- includes array processing operations similar to NumPy, automatic differentiation, a parallelization-centric just-in-time compiler, and automated integration with hardware accelerators such as GPUs.

Pandas

- provides types and functions for manipulating data.

Numba

- provides a just-in-time compiler that plays well with NumPy and helps accelerate Python code.

Numpy (Numerical Python) Library

```
import numpy as np
```

NumPy is a first-rate library for numerical programming, and scientific computing

- Widely used in academia, finance and industry.
- Mature, fast, stable and under continuous development.
- Work with **Tensorflow** and **Pytorch library**

1D array

- shape: (axis 0,)
- note: **axis 0**: number of element

2D array

- shape: (axis 0, axis 1)
- note: **axis 0**: number of rows; **axis 1**: number of columns

3D array

- shape: (axis 0, axis 1, axis 2)
- note: **axis 0**: number of 2D arrays; **axis 1**: number of rows of 2D array; **axis 2**: number of comlums of 2D array

shape

- return a tuple of number elements in each dimension

ndim

- return number of dimension

dtype

- data type for array

```
from numpy import *

data = array([5.23 ,5.9])
print(data,'\n')

# return a tuple of number elements in each dimension
print(data.shape,'\n')

# return number of dimension
print(data.ndim, '\n')

# return data type
print(data.dtype)
```

```
[5.23 5.9 ]

(2,)

1

float64
```

```
from numpy import *

data = array([[5,5,8,9],[0,4,2,7],[7,9,10,2]])
print(data,'\n')

# return a tuple of number elements in each dimension
print(data.shape,'\n')

# return number of dimension
print(data.ndim, '\n')

# return data type
print(data.dtype)

#update data[2,3]
data[2,3] = 999

print(data)
```

```
[[ 5  5  8  9]
 [ 0  4  2  7]
 [ 7  9 10  2]]

(3, 4)
```

```
2

int64
[[   5    5    8    9]
 [   0    4    2    7]
 [   7    9   10  999]]
```

Data type

- **Boolean**: bool
- **Integer**: int8, int16, int32, int64, int128, int
- **Unsigned Integer**: uint8, uint16, uint32, uint64, uint128, uint
- **Float**: float32, float64, float, longfloat,
- **Complex**: complex64, complex128, complex
- **Strings**: str, unicode
- **Object**: object
- **Records**: void

Slicing

- 1D array

```
a[x:y:z]
# x: begin
# y: end
# z: step
```

- 2D array

```
a[x1:y1:z1,x2:y2:z2]
#1: row
#2: column
```

- 3D array

```
a[x1:y1:z1,x2:y2:z2,x3,y3,z3]
#1: array
#2: row
#3: column

from numpy import *

a = array([[0,1,2,3,4,5],
           [10,11,12,13,14,15],
           [20,21,22,23,24,25],
           [30,31,32,33,34,35],
           [40,41,42,43,44,45],
           [50,52,52,53,54,55]])
print(a,'\n')
print(a[0,3:4],'\n')
print(a[4:,4:],'\n')
print(a[:,2],'\n')
print(a[2::2,::2])
```

```
[[ 0  1  2  3  4  5]
 [10 11 12 13 14 15]
 [20 21 22 23 24 25]
 [30 31 32 33 34 35]
 [40 41 42 43 44 45]
 [50 52 52 53 54 55]]

[3]

[[44 45]
 [54 55]]

[ 2 12 22 32 42 52]

[[20 22 24]
 [40 42 44]]
```

Create array

- arange(x,y,z): create an 1D array **start** at x, **end** at y, **step**: z
- eye(x): create a 2D array with ones on a specified diagonal and zeros elsewhere
- random():
- ones(): create a 2D array with ones

```
from numpy import *

arr1 = arange(5)
print(f"arr1: \n{arr1}",'\n')

arr2 = arange(0, 5, 2)
print(f"arr2: \n{arr2}",'\n')

arr3 = eye(3)
print(f"arr3: \n{arr3}",'\n')

arr4 = random.rand(3,2)
print(f"arr4: \n{arr4} \n")

arr5 = ones((3,2))
print(f"arr5: \n{arr5} \n")

arr6 = ones(3)
print(f"arr5: \n{arr6} \n")
```

```
arr1:
[0 1 2 3 4]
```

```
arr2:
[0 2 4]

arr3:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

arr4:
[[0.6793149  0.5736319 ]
 [0.57150994 0.5923853 ]
 [0.72530854 0.65532766]]

arr5:
[[1. 1.]
 [1. 1.]
 [1. 1.]]

arr5:
[1. 1. 1.]
```

Horrizontal Concatenate

```
from numpy import *

arr1 = array([1,2,3])
arr2 = array([4,5,6])
arr3 = concatenate([arr1, arr2])
#similar to
arr4 = concatenate((arr1, arr2), axis = 0)
print(arr3,'\n')
print(arr4)
```

```
[1 2 3 4 5 6]
[1 2 3 4 5 6]
```

Vertical Concatenate

- 2 array must have the same number of column

```
from numpy import *

arr1 = array([[1,2,3]])
arr2 = array([[4,5,6]])
arr3 = concatenate([arr1, arr2])
#similar to
arr4 = concatenate((arr1, arr2), axis = 0)
print(arr3,'\n')
print(arr4,'\n')
```

```
arr5 = concatenate((arr1, arr2), axis = 1)
print(arr5,'\n')
```

```
[[1 2 3]
 [4 5 6]]

[[1 2 3]
 [4 5 6]]

[[1 2 3 4 5 6]]
```

Statistics

- sum()
- mean()
- std()
- median()

```
from numpy import *

a = array([[0,1,2,3,4,5],
           [10,11,12,13,14,15],
           [20,21,22,23,24,25],
           [30,31,32,33,34,35],
           [40,41,42,43,44,45],
           [50,52,52,53,54,55]])
print(f"sum: {sum(a)}")
print(f"mean: {mean(a)}")
print(f"std: {std(a)}")
print(f"median: {median(a)}")
```

```
sum: 991
mean: 27.52777777777778
std: 17.20220611031412
median: 27.5
```

Arithmetic Opetations

+

-

*

/

**

@

```
from numpy import *
a = array([1,2,3,4])
b = array([5,6,7,8])

print(f"a+b: {a+b}")
print(f"a-b: {a-b}")
print(f"a*b: {a*b}")
print(f"a/b: {a/b}")
print(f"a**b: {a**b}")
print(f"a@b: {a@b}")
```

```
a+b: [ 6  8 10 12]
a-b: [-4 -4 -4 -4]
a*b: [ 5 12 21 32]
a/b: [0.2        0.33333333 0.42857143 0.5       ]
a**b: [    1    64  2187 65536]
a@b: 70
```

```
from numpy import *
a = array([[1,2],[3,4]])
b = array([[2],[1]])
c = array([2,1])

print(f"a+b: \n{a+b}\n")
print(f"a-b: \n{a-b}\n")
print(f"a*b: \n{a*b}\n")
print(f"a/b: \n{a/b}\n")
print(f"a**b: \n{a**b}\n")
print(f"a@b: \n{a@b}\n")
print(f"a@c: \n{a@c}\n")
```

```
a+b:
[[3 4]
 [4 5]]

a-b:
[[-1  0]
 [ 2  3]]

a*b:
[[2 4]
 [3 4]]

a/b:
[[0.5 1. ]
 [3.  4. ]]

a**b:
[[1 4]
 [3 4]]
```

```
a@b:
[[ 4]
 [10]]

a@c:
[ 4 10]
```

We **can also use +, -, *, /, **, @** when one element is a Python **list or tuple**

The tuple `(x,)` is treated as a column vector `(,x)`

```
from numpy import *
a = array(((1,2),(3,4)))
b = array((2,1))

print(f"a+b: \n{a+b}\n")
print(f"a-b: \n{a-b}\n")
print(f"a*b: \n{a*b}\n")
print(f"a/b: \n{a/b}\n")
print(f"a**b: \n{a**b}\n")
print(f"a@b: \n{a@b}\n")

a+b:
[[3 3]
 [5 5]]

a-b:
[[-1  1]
 [ 1  3]]

a*b:
[[2 2]
 [6 4]]

a/b:
[[0.5 2. ]
 [1.5 4. ]]

a**b:
[[1 2]
 [9 4]]

a@b:
[ 4 10]
```

Pandas

- defines fundamental structures for working with data
- endows them with methods that facilitate operations such as

- reading in data
- adjusting indices
- working with dates and time series
- sorting, grouping, re-ordering and general data munging
- dealing with missing values, etc.,

Series/Variable

- 1: "column" ---> 1 variable

Data Frames

- several columns ---> serveral variables
- similar to **Excel spreadsheet**
- powerful tool for presenting and analyzing data that are naturally organized into rows and columns

Some NumPy functions

- `describe(var_x)` or `var_x.describe()`: descriptive statistic
- `abs(var_x)` or `var_x.abs()`
- `nan`: not a number, data type: float. e.g. `df.loc[df.total_bill == min(df.total_bill), 'total_bill'] = nan` fill nan into the total_bill if total_bill = min(total_bill)
- `round(x,y)`: round x to y decimal places

Some Pandas functions

- `df = read_csv('link')`
- `df[['year', 'POP', 'XRAT', 'tcgdp', 'cc', 'cg']].apply(max)`: find max value of these variables
- `map()`: modify all individual entries in the dataframe altogether
- `df = df.set_index('var_a')`: set the index to the the var_a in the dataframe

```
from pandas import *
from numpy import *
from matplotlib.pyplot import *
from requests import *

#create a series of 4 random observations
s = Series(random.rand(4), name = 'daily returns')
b = Series([-1.5, 2.3, -3.4, 4.5], name = 'daily returns')
print(b,'\n')

b.rt = [0.05, 0.1, 0.15, 0.2]
b.index = ['AMZN', 'GOOGL', 'MSFT', 'AAPL']
print(b,'\n')

'ASSS' in b
```

```
print(b.rt)
print(b.index,'\n')
print(b.describe(),'\n')


0   -1.5
1    2.3
2   -3.4
3    4.5
Name: daily returns, dtype: float64

AMZN    -1.5
GOOGL    2.3
MSFT    -3.4
AAPL     4.5
Name: daily returns, dtype: float64

[0.05, 0.1, 0.15, 0.2]
Index(['AMZN', 'GOOGL', 'MSFT', 'AAPL'], dtype='object')

count    4.000000
mean     0.475000
std      3.579921
min     -3.400000
25%     -1.975000
50%      0.400000
75%      2.850000
max      4.500000
Name: daily returns, dtype: float64
```

Select data

- we have a dataframe: df
- `df[x:n]`: select data from row x to row n
- `df.iloc[x:n, y:m]`: select data from row x to row n, column y to column m
- `df.loc[df.index[x:n], ['var_a','var_b']]`: select data from row x to row n, columns/variables var_a and var_b

Select data with condition

- `df.loc[df.cc == max(df.cc)]`
- `df.loc[(df.cc + df.cg >= 80) & (df.POP <= 20000), ['country', 'year', 'POP']]`
- `df[df.var_a >= 20000]` or `df.query(var_a >=20000)`: select observations if var_a >= 20000
- `df[(df.country.isin(['Argentina','India','South Africa'])) & (df.POP > 40000)]` or `df.query("country in ['Argentina','India','South Africa'] and POP > 40000)`: select observations if (country == (Argentina, India, or South Africa) & POP > 40000)

- `df[(df.cc + df.cg >= 80) & (df.POP <= 20000)]` or `df.query("cc+cg >= 80 & POP <= 20000")`

Select data with advanced condition

- `df[['var_a','var_b','var_c']].apply(max)`: find max of var_a, var_b, var_c
- `df.apply(lambda row: row, axis = 1)`: return itself for each row in the dataframe
- `df.apply(lambda row: row['tip'] > 5 and row['size'] >= 3, axis=1)`: return True if (tip > 5 & size >= 3), else False
- `df[complexCondition]` similar to `df.loc[complexCondition]`: return rows if complexCondition == True

```
df = read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv')
print(df,'\n')
print(df[2:5],'\n')
print(df[['total_bill','day']],'\n')
print(df.iloc[3:8, 0:3],'\n')
print(df[(df.total_bill + df.tip >= 20) & (df.sex == 'Female')])

     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]

   total_bill   tip     sex smoker  day    time  size
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4

     total_bill   day
0         16.99   Sun
1         10.34   Sun
2         21.01   Sun
3         23.68   Sun
4         24.59   Sun
..          ...   ...
239       29.03   Sat
240       27.18   Sat
```

```
241       22.67   Sat
242       17.82   Sat
243       18.78  Thur

[244 rows x 2 columns]

   total_bill   tip     sex
3       23.68  3.31    Male
4       24.59  3.61  Female
5       25.29  4.71    Male
6        8.77  2.00    Male
7       26.88  3.12    Male

     total_bill   tip     sex smoker   day    time  size
4         24.59  3.61  Female     No   Sun  Dinner     4
11        35.26  5.00  Female     No   Sun  Dinner     4
18        16.97  3.50  Female     No   Sun  Dinner     3
21        20.29  2.75  Female     No   Sat  Dinner     2
29        19.65  3.00  Female     No   Sat  Dinner     2
33        20.69  2.45  Female     No   Sat  Dinner     4
37        16.93  3.07  Female     No   Sat  Dinner     3
52        34.81  5.20  Female     No   Sun  Dinner     4
57        26.41  1.50  Female     No   Sat  Dinner     2
71        17.07  3.00  Female     No   Sat  Dinner     3
72        26.86  3.14  Female    Yes   Sat  Dinner     2
73        25.28  5.00  Female    Yes   Sat  Dinner     2
85        34.83  5.17  Female     No  Thur   Lunch     4
93        16.32  4.30  Female    Yes   Fri  Dinner     2
94        22.75  3.25  Female     No   Fri  Dinner     2
102       44.30  2.50  Female    Yes   Sat  Dinner     3
103       22.42  3.48  Female    Yes   Sat  Dinner     2
104       20.92  4.08  Female     No   Sat  Dinner     2
114       25.71  4.00  Female     No   Sun  Dinner     3
115       17.31  3.50  Female     No   Sun  Dinner     2
119       24.08  2.92  Female     No  Thur   Lunch     4
125       29.80  4.20  Female     No  Thur   Lunch     6
131       20.27  2.83  Female     No  Thur   Lunch     2
134       18.26  3.25  Female     No  Thur   Lunch     2
140       17.47  3.50  Female     No  Thur   Lunch     2
143       27.05  5.00  Female     No  Thur   Lunch     6
146       18.64  1.36  Female     No  Thur   Lunch     3
155       29.85  5.14  Female     No   Sun  Dinner     5
157       25.00  3.75  Female     No   Sun  Dinner     4
164       17.51  3.00  Female    Yes   Sun  Dinner     2
186       20.90  3.50  Female    Yes   Sun  Dinner     3
188       18.15  3.50  Female    Yes   Sun  Dinner     3
191       19.81  4.19  Female    Yes  Thur   Lunch     2
197       43.11  5.00  Female    Yes  Thur   Lunch     4
214       28.17  6.50  Female    Yes   Sat  Dinner     3
219       30.14  3.09  Female    Yes   Sat  Dinner     4
```

```
229        22.12  2.88  Female    Yes   Sat  Dinner     2
238        35.83  4.67  Female     No   Sat  Dinner     3
240        27.18  2.00  Female    Yes   Sat  Dinner     2
243        18.78  3.00  Female     No  Thur  Dinner     2
```

Apply `apply()`, `lambda`, `read_csv()`

```
df = read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv')
print(df[['total_bill','tip', 'size']].apply(max), '\n')

print(df.apply(lambda row: row, axis = 1), '\n')

df['bill_with_tax'] = df['total_bill'].apply(lambda x: x * 1.1)
print(df[0:5], '\n')

total_bill    50.81
tip           10.00
size           6.00
dtype: float64

     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]

   total_bill   tip     sex smoker  day    time  size  bill_with_tax
0       16.99  1.01  Female     No  Sun  Dinner     2         18.689
1       10.34  1.66    Male     No  Sun  Dinner     3         11.374
2       21.01  3.50    Male     No  Sun  Dinner     3         23.111
3       23.68  3.31    Male     No  Sun  Dinner     2         26.048
4       24.59  3.61  Female     No  Sun  Dinner     4         27.049
```

Complex Condition

```
df = read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv')
complexCondition = df.apply(lambda row: row['tip'] > 5 and row['size']
>= 3, axis=1)
print(df[complexCondition], '\n')
print("test")
print(df.loc[complexCondition], '\n')
print("test",complexCondition, '\n')
```

```
     total_bill    tip     sex smoker   day    time  size
23        39.42   7.58    Male     No   Sat  Dinner     4
44        30.40   5.60    Male     No   Sun  Dinner     4
47        32.40   6.00    Male     No   Sun  Dinner     4
52        34.81   5.20  Female     No   Sun  Dinner     4
59        48.27   6.73    Male     No   Sat  Dinner     4
85        34.83   5.17  Female     No  Thur   Lunch     4
116       29.93   5.07    Male     No   Sun  Dinner     4
141       34.30   6.70    Male     No  Thur   Lunch     6
155       29.85   5.14  Female     No   Sun  Dinner     5
170       50.81  10.00    Male    Yes   Sat  Dinner     3
183       23.17   6.50    Male    Yes   Sun  Dinner     4
211       25.89   5.16    Male    Yes   Sat  Dinner     4
212       48.33   9.00    Male     No   Sat  Dinner     4
214       28.17   6.50  Female    Yes   Sat  Dinner     3
239       29.03   5.92    Male     No   Sat  Dinner     3

test
     total_bill    tip     sex smoker   day    time  size
23        39.42   7.58    Male     No   Sat  Dinner     4
44        30.40   5.60    Male     No   Sun  Dinner     4
47        32.40   6.00    Male     No   Sun  Dinner     4
52        34.81   5.20  Female     No   Sun  Dinner     4
59        48.27   6.73    Male     No   Sat  Dinner     4
85        34.83   5.17  Female     No  Thur   Lunch     4
116       29.93   5.07    Male     No   Sun  Dinner     4
141       34.30   6.70    Male     No  Thur   Lunch     6
155       29.85   5.14  Female     No   Sun  Dinner     5
170       50.81  10.00    Male    Yes   Sat  Dinner     3
183       23.17   6.50    Male    Yes   Sun  Dinner     4
211       25.89   5.16    Male    Yes   Sat  Dinner     4
212       48.33   9.00    Male     No   Sat  Dinner     4
214       28.17   6.50  Female    Yes   Sat  Dinner     3
239       29.03   5.92    Male     No   Sat  Dinner     3

test 0      False
1      False
2      False
3      False
4      False
       ...
239     True
240    False
241    False
```

```
242    False
243    False
Length: 244, dtype: bool
```

Fill missing value

Apply `round(), nan, isnan(), map()`

```
df = read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/tips.csv')
df.loc[df.total_bill == min(df.total_bill), 'total_bill'] = nan
df.loc[df.total_bill == max(df.total_bill), 'total_bill'] = nan
print(df[df.total_bill.isna()], '\n')

# df[df.total_bill.isna()] = 0 ---> không dùng cái này vì fill số 0 cả dòng
df.loc[df.total_bill.isna(), 'total_bill'] = 0

print(df.loc[df['total_bill'] == 0, 'total_bill'], '\n')

print(df[df.total_bill == 0], '\n')

df = df.apply(lambda x: round(x,1) if x.dtype != 'str' else x)
# similar to
# df = df.map(lambda x: round(x, 1) if isinstance(x, (int, float)) else x)
# df.map(lambda x: round(x,1) if type(x) != str else x)
print(df[0:3],'\n')

     total_bill   tip     sex smoker  day    time  size
67          NaN   1.0  Female    Yes  Sat  Dinner     1
170         NaN  10.0    Male    Yes  Sat  Dinner     3

67     0.0
170    0.0
Name: total_bill, dtype: float64

     total_bill   tip     sex smoker  day    time  size
67          0.0   1.0  Female    Yes  Sat  Dinner     1
170         0.0  10.0    Male    Yes  Sat  Dinner     3

   total_bill  tip     sex smoker  day    time  size
0        17.0  1.0  Female     No  Sun  Dinner     2
1        10.3  1.7    Male     No  Sun  Dinner     3
2        21.0  3.5    Male     No  Sun  Dinner     3
```

Replace value using `list` and `zip`

```
df = read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv')
for index in list(zip([0,3,5,6],[3,4,6,2])):
    df.iloc[index] = nan

print(df,'\n')

df = df.fillna(0)
print(df,'\n')

     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female    NaN   Sun  Dinner   2.0
1         10.34  1.66    Male     No   Sun  Dinner   3.0
2         21.01  3.50    Male     No   Sun  Dinner   3.0
3         23.68  3.31    Male     No   NaN  Dinner   2.0
4         24.59  3.61  Female     No   Sun  Dinner   4.0
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner   3.0
240       27.18  2.00  Female    Yes   Sat  Dinner   2.0
241       22.67  2.00    Male    Yes   Sat  Dinner   2.0
242       17.82  1.75    Male     No   Sat  Dinner   2.0
243       18.78  3.00  Female     No  Thur  Dinner   2.0

[244 rows x 7 columns]

     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female      0   Sun  Dinner   2.0
1         10.34  1.66    Male     No   Sun  Dinner   3.0
2         21.01  3.50    Male     No   Sun  Dinner   3.0
3         23.68  3.31    Male     No     0  Dinner   2.0
4         24.59  3.61  Female     No   Sun  Dinner   4.0
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner   3.0
240       27.18  2.00  Female    Yes   Sat  Dinner   2.0
241       22.67  2.00    Male    Yes   Sat  Dinner   2.0
242       17.82  1.75    Male     No   Sat  Dinner   2.0
243       18.78  3.00  Female     No  Thur  Dinner   2.0

[244 rows x 7 columns]


df = read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv')
#write a function to replace nan values
def replace_nan(x):
    if type(x) != str and isnan(x):
        return 0
    else: return x

for index in list(zip([0,3,5,6],[3,4,6,2])):
    df.iloc[index] = nan

print(df,'\n')

df.map(replace_nan)

     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female    NaN   Sun  Dinner   2.0
1         10.34  1.66    Male     No   Sun  Dinner   3.0
2         21.01  3.50    Male     No   Sun  Dinner   3.0
3         23.68  3.31    Male     No   NaN  Dinner   2.0
4         24.59  3.61  Female     No   Sun  Dinner   4.0
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner   3.0
240       27.18  2.00  Female    Yes   Sat  Dinner   2.0
241       22.67  2.00    Male    Yes   Sat  Dinner   2.0
242       17.82  1.75    Male     No   Sat  Dinner   2.0
243       18.78  3.00  Female     No  Thur  Dinner   2.0

[244 rows x 7 columns]


     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female      0   Sun  Dinner   2.0
1         10.34  1.66    Male     No   Sun  Dinner   3.0
2         21.01  3.50    Male     No   Sun  Dinner   3.0
3         23.68  3.31    Male     No     0  Dinner   2.0
4         24.59  3.61  Female     No   Sun  Dinner   4.0
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner   3.0
240       27.18  2.00  Female    Yes   Sat  Dinner   2.0
241       22.67  2.00    Male    Yes   Sat  Dinner   2.0
242       17.82  1.75    Male     No   Sat  Dinner   2.0
243       18.78  3.00  Female     No  Thur  Dinner   2.0

[244 rows x 7 columns]

df = read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv')
for index in list(zip([0,3,1,6],[3,0,1,2])):
    df.iloc[index] = nan

print(df,'\n')

df = df.fillna(df.iloc[:,0:2].mean())
df = df.apply(lambda x: round(x,1) if x.dtype != 'str' else x)

print(df,'\n')

     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female    NaN   Sun  Dinner     2
```

```
1         10.34   NaN    Male      No   Sun  Dinner     3
2         21.01  3.50    Male      No   Sun  Dinner     3
3           NaN  3.31    Male      No   Sun  Dinner     2
4         24.59  3.61  Female      No   Sun  Dinner     4
..          ...   ...     ...     ...   ...     ...   ...
239       29.03  5.92    Male      No   Sat  Dinner     3
240       27.18  2.00  Female     Yes   Sat  Dinner     2
241       22.67  2.00    Male     Yes   Sat  Dinner     2
242       17.82  1.75    Male      No   Sat  Dinner     2
243       18.78  3.00  Female      No  Thur  Dinner     2

[244 rows x 7 columns]

     total_bill  tip     sex smoker   day    time  size
0          17.0  1.0  Female    NaN   Sun  Dinner     2
1          10.3  3.0    Male     No   Sun  Dinner     3
2          21.0  3.5    Male     No   Sun  Dinner     3
3          19.8  3.3    Male     No   Sun  Dinner     2
4          24.6  3.6  Female     No   Sun  Dinner     4
..          ...  ...     ...    ...   ...     ...   ...
239        29.0  5.9    Male     No   Sat  Dinner     3
240        27.2  2.0  Female    Yes   Sat  Dinner     2
241        22.7  2.0    Male    Yes   Sat  Dinner     2
242        17.8  1.8    Male     No   Sat  Dinner     2
243        18.8  3.0  Female     No  Thur  Dinner     2

[244 rows x 7 columns]
```

Standardization and Visualization

```python
df = read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv')
# we are only interest in the total_bill, sex, and smoker
df = df[['total_bill','tip','sex','smoker']]

# create a date range
df['day'] = date_range(start='2023-01-02', periods=len(df), freq='D')

# set the day as the index
df = df.set_index('day')

# you can also change name of variables
df.columns = 'total', 'bonus', 'gender', 'smoker'

print(df)

# transform variable
df['total'] = df['total'] * 1.1  # add 10% tax
```

```
df

            total  bonus  gender smoker
day
2023-01-02  16.99   1.01  Female     No
2023-01-03  10.34   1.66    Male     No
2023-01-04  21.01   3.50    Male     No
2023-01-05  23.68   3.31    Male     No
2023-01-06  24.59   3.61  Female     No
...           ...    ...     ...    ...
2023-08-29  29.03   5.92    Male     No
2023-08-30  27.18   2.00  Female    Yes
2023-08-31  22.67   2.00    Male    Yes
2023-09-01  17.82   1.75    Male     No
2023-09-02  18.78   3.00  Female     No

[244 rows x 4 columns]

             total  bonus  gender smoker
day
2023-01-02  18.689   1.01  Female     No
2023-01-03  11.374   1.66    Male     No
2023-01-04  23.111   3.50    Male     No
2023-01-05  26.048   3.31    Male     No
2023-01-06  27.049   3.61  Female     No
...            ...    ...     ...    ...
2023-08-29  31.933   5.92    Male     No
2023-08-30  29.898   2.00  Female    Yes
2023-08-31  24.937   2.00    Male    Yes
2023-09-01  19.602   1.75    Male     No
2023-09-02  20.658   3.00  Female     No

[244 rows x 4 columns]
```