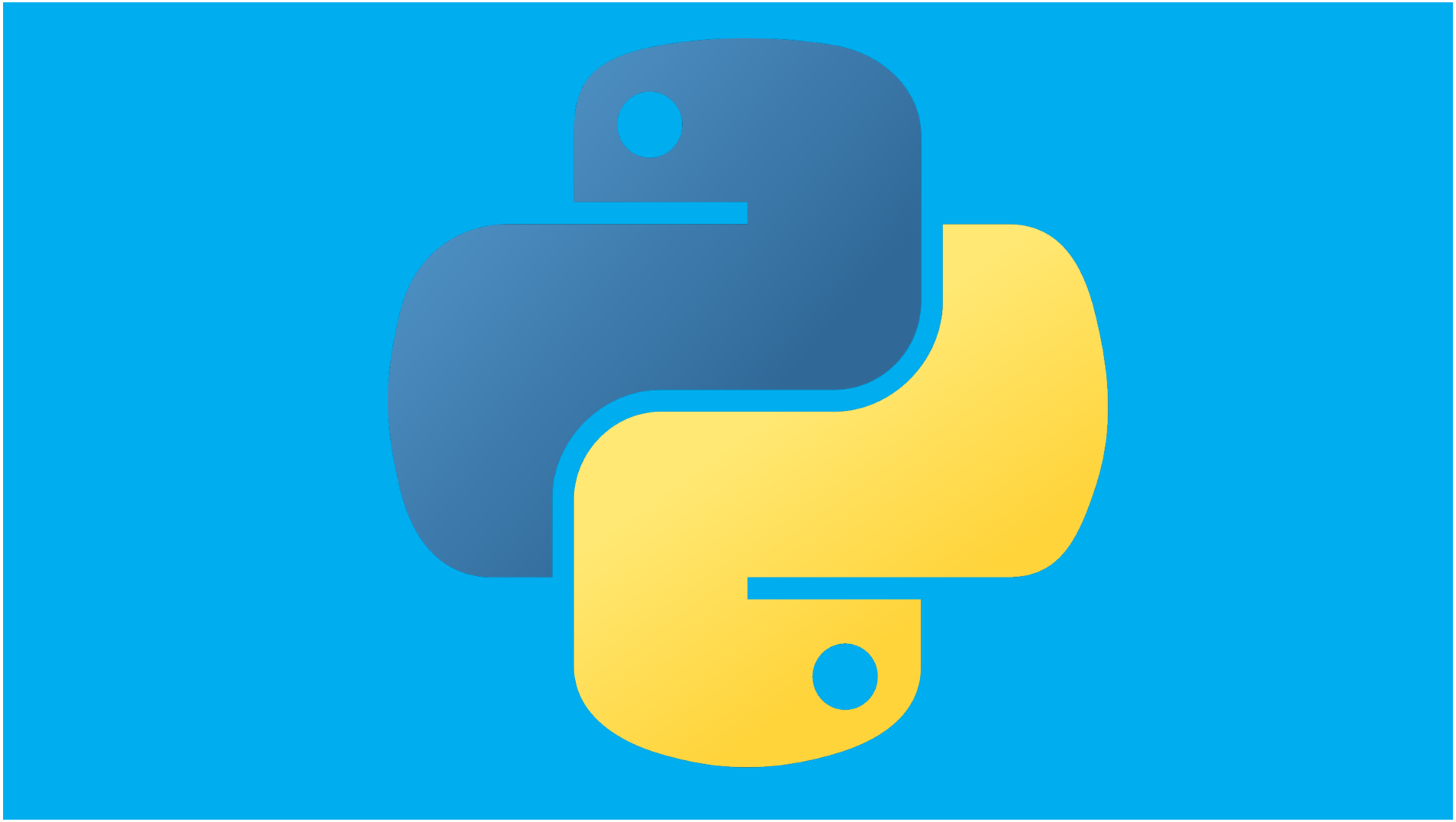
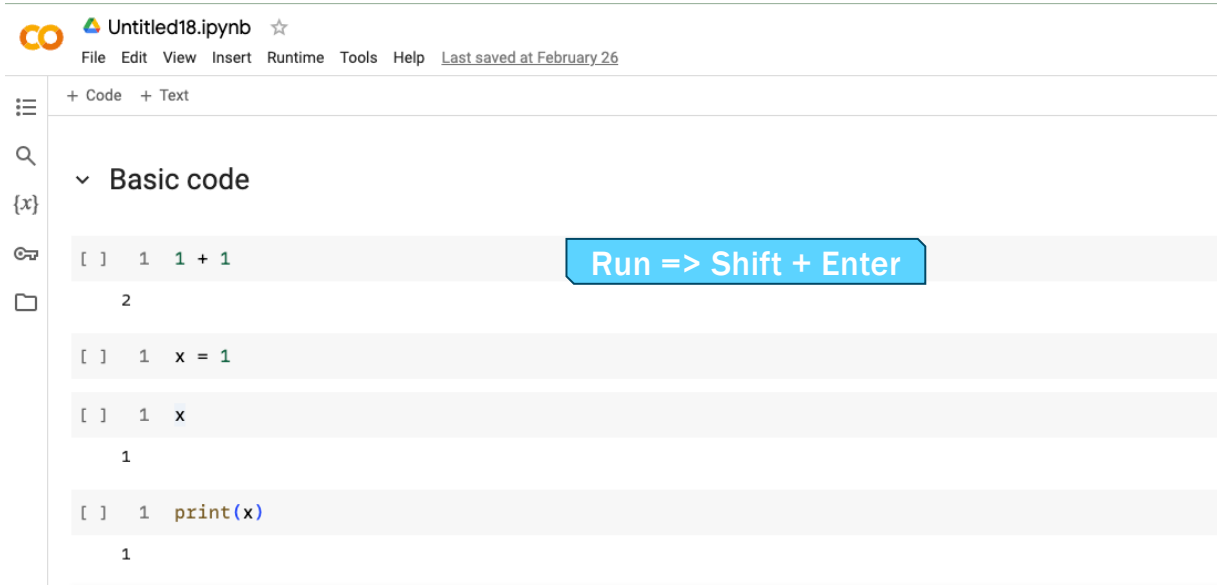


Colab (online)



https://colab.research.google.com/

Outlines

- Data types
  - Python data types
  - Numpy
- Pandas
  - Import data
  - Pandas plot
  - Seaborn plot
- Routing
  - Find distance matrix
  - Create a routing solution

Outlines

- Data types (numeric)
  - Integer ex. -1
  - Real ex. 1.0
  - Complex ex. 1+2j
- Datatypes (sequence)
  - String ex. ‘hello’
  - List ex. [0, 1.1, 2, ‘x’, ‘abc’, 1-2j, [0, 1], (2, 1)]
  - Tuple ex. (0, 1.1, 2, ‘x’, ‘abc’, 1-2j, [0, 1], (2, 1))
  - Set ex. {0, 1.1, 2, ‘x’, ‘abc’, 1-2j, [0, 1], (2, 1)}
  - Dictionary ex. {1: ‘test’, “x”: 4, 1.2: 3.4, (0, 1): (2, 1)}

Data types (str: string)

```
[ ]  1  x = 'hello world'
     2  print(x)
```

```
hello world
```

```
[ ]  1  x = 'hello world'
     2  x
```

```
'hello world'
```

```
[ ]  1  x = "hello world"
     2  print(x)
```

```
hello world
```

```
[ ]  1  x = "hello world"
     2  x
```

```
'hello world'
```

```
[ ]  1  type(x)
```

```
str
```

Data types

int (integer)

```
[11]  x = 10
```

```
[12]  x
```

```
10
```

```
[13]  x = 10
      print(x)
```

```
10
```

```
[14]  type(x)
```

```
int
```

float (real)

```
[15]  x = 10.321
```

```
[16]  x
```

```
10.321
```

```
[17]  x = 10.321
      print(x)
```

```
10.321
```

```
[18]  type(x)
```

```
float
```

complex

```
[19]  x = 1 + 3j
```

```
[21]  x
```

```
(1+3j)
```

```
[22]  print(x)
```

```
(1+3j)
```

```
[20]  type(x)
```

```
complex
```

Comment code

```
a = 1
b = 2
c = 3
```

Ctrl+/

```
# a = 1
# b = 2
# c = 3
```

Data types (list)

list = collection of data

```
X = [1, 2, 3, 4.5, 'hello', "world"]
      ↑   ↑   ↑   ↑      ↑        ↑
Index:  0   1   2   3      4        5

Index: -6  -5  -4  -3     -2       -1
```

	Python	MATLAB
First element	`X[0] = 1`	`X(1)`
Last element	`X[-1] = 'world'`	`X(end)`

Data types (list)

Slicing in list

```
x[Start:Stop:Step]
```

```
        X = [1, 2, 3, 4.5, 'hello', "world"]
             ↑   ↑   ↑   ↑      ↑       ↑
Index:       0   1   2   3      4       5

Index:      -6  -5  -4  -3     -2      -1
```

	Python	MATLAB
First element	`X[0] = 1`	`X(1)`
Last element	`X[-1] = 'world'`	`X(end)`
Element 1, 3, 5	`X[0:5:2], x[:5:2] = [1, 3, 'hello']`	`X(1:2:5)`
First 3 elements	`X[:3] = [1, 2, 3]`	`X(1:3)`

Exercises:

1. **X[?:?:?]** = [2, 4.5, 'world']
2. **X[?:?:?]** = [3, 4.5, 'hello']
3. Reverse of x => **X[?:?:?]** = ['world', 'hello', 4.5, 3, 2, 1]

Data types (list)

Change value in list

```
        X = [1, 2, 3, 4.5, 'hello', "world"]
             ↑  ↑  ↑  ↑      ↑        ↑
Index:       0  1  2  3      4        5

Index:      -6 -5 -4 -3     -2       -1
```

```
[14]  1  X = [1, 2, 3, 4.5, 'hello', "world"]
      2  X
```

```
[1, 2, 3, 4.5, 'hello', 'world']
```

```
[15]  1  X[5] = 'You'
```

เปลี่ยนให้ตำแหน่งที่ 5 ของ X มีค่าเป็น 'You'

```
[16]  1  X
```

```
[1, 2, 3, 4.5, 'hello', 'You']
```

Exercises:

1. X = ['123', 2, 3, 4.5, 'hello', 'You']

Data types (list)

Slicing in list (string)

```
[10]  x = 'hello world!'
```

```
[11]  x[0]
```

```
'h'
```

```
[12]  x[-2]
```

```
'd'
```

```
[13]  x[:-2]
```

```
'hello worl'
```

F-string

Format string

```
[43]  f'x = {x}'
```

```
'x = hello world!'
```

```
[44]  f'hello {x:=^30}'
```

```
'hello =========hello world!=========='
```

```
[45]  balance = 53234.34563345
      f"Balance: ${balance:.2f}"
```

```
'Balance: $53234.35'
```

```
[46]  company_value = 1405940204920
      print(f"The value of the company is {company_value:,d}")
```

```
The value of the company is 1,405,940,204,920
```

Built-in Functions

range(*start*, *stop*, *step*) = [start, start+step, start+2step,…,stop-1]

```
1  range(10)
```

```
range(0, 10)
```

```
1  range(1, 10)
```

```
range(1, 10)
```

```
1  list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
1  list(range(1, 10))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[2]  1  list(range(3, 20, 5))
```

```
[3, 8, 13, 18]
```

```
[6]  1  list(range(-10, 20, 6))
```

```
[-10, -4, 2, 8, 14]
```

```
[7]  1  list(range(1.2, 10.2, 0.2))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-ccf9cacace7c> in <cell line: 1>()
----> 1 list(range(1.2, 10.2, 0.2))

TypeError: 'float' object cannot be interpreted as an integer
```

For loop

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[115]
```
for i in range(10):
    print(f'i = {i}')
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
```

```
for a in [1, 2, 3, 4.5, 'hello', "world"]
    print(f'a = {a}')
```

```
a = 1
a = 2
a = 3
a = 4.5
a = hello
a = world
```

[129]
```
for i in range(5):
    for j in range(5):
        print(f'i = {i:2d}  j = {j:2d}')
```

```
i =  0  j =  0
i =  0  j =  1
i =  0  j =  2
i =  0  j =  3
i =  0  j =  4
i =  1  j =  0
i =  1  j =  1
i =  1  j =  2
i =  1  j =  3
i =  1  j =  4
i =  2  j =  0
i =  2  j =  1
i =  2  j =  2
i =  2  j =  3
i =  2  j =  4
i =  3  j =  0
i =  3  j =  1
i =  3  j =  2
i =  3  j =  3
i =  3  j =  4
i =  4  j =  0
i =  4  j =  1
i =  4  j =  2
i =  4  j =  3
i =  4  j =  4
```

Exercise:

```
for ? in range(5):
    for ? in range(5):
        print(f'???')
```

```
0 + 0 = 0
0 + 1 = 1
0 + 2 = 2
0 + 3 = 3
0 + 4 = 4
1 + 0 = 1
1 + 1 = 2
1 + 2 = 3
1 + 3 = 4
1 + 4 = 5
2 + 0 = 2
2 + 1 = 3
2 + 2 = 4
2 + 3 = 5
2 + 4 = 6
3 + 0 = 3
3 + 1 = 4
3 + 2 = 5
3 + 3 = 6
3 + 4 = 7
4 + 0 = 4
4 + 1 = 5
4 + 2 = 6
4 + 3 = 7
4 + 4 = 8
```

If else

```
x = 15

if x/2 <= 8:
    print(f'x < 16')
elif x > 3:
    print(f'x > 3')
else:
    print(f'x is less than 3')
```

```
x > 3
```

Exercise

```
x = ???

if x/2 <= 8:
    print(f'x < 16')
elif x > 3:
    print(f'x > 3')
else:
    print(f'x is less than 3')
```

```
x < 16
```

```
for i in range(5):
    for j in range(5):
        print(f'{i} + {j} = {i+j}')
```

i+j <= 5

```
for i in range(5):
    for j in range(5):
        if i + j <= 5:
            print(f'{i} + {j} = {i+j}')
```

```
0 + 0 = 0
0 + 1 = 1
0 + 2 = 2
0 + 3 = 3
0 + 4 = 4
1 + 0 = 1
1 + 1 = 2
1 + 2 = 3
1 + 3 = 4
1 + 4 = 5
2 + 0 = 2
2 + 1 = 3
2 + 2 = 4
2 + 3 = 5
3 + 0 = 3
3 + 1 = 4
3 + 2 = 5
4 + 0 = 4
4 + 1 = 5
```

Exercise

(i+j+k <= 5)

```
0 + 0 + 0 = 0
0 + 0 + 1 = 1
0 + 0 + 2 = 2
0 + 0 + 3 = 3
0 + 0 + 4 = 4
0 + 1 + 0 = 1
0 + 1 + 1 = 2
0 + 1 + 2 = 3
0 + 1 + 3 = 4
0 + 1 + 4 = 5
0 + 2 + 0 = 2
0 + 2 + 1 = 3
0 + 2 + 2 = 4
0 + 2 + 3 = 5
0 + 3 + 0 = 3
0 + 3 + 1 = 4
0 + 3 + 2 = 5
0 + 4 + 0 = 4
0 + 4 + 1 = 5
1 + 0 + 0 = 1
1 + 0 + 1 = 2
1 + 0 + 2 = 3
1 + 0 + 3 = 4
1 + 0 + 4 = 5
1 + 1 + 0 = 2
1 + 1 + 1 = 3
1 + 1 + 2 = 4
1 + 1 + 3 = 5
1 + 2 + 0 = 3
1 + 2 + 1 = 4
1 + 2 + 2 = 5
1 + 3 + 0 = 4
1 + 3 + 1 = 5
1 + 4 + 0 = 5
2 + 0 + 0 = 2
2 + 0 + 1 = 3
2 + 0 + 2 = 4
2 + 0 + 3 = 5
2 + 1 + 0 = 3
2 + 1 + 1 = 4
2 + 1 + 2 = 5
2 + 2 + 0 = 4
2 + 2 + 1 = 5
2 + 3 + 0 = 5
3 + 0 + 0 = 3
3 + 0 + 1 = 4
3 + 0 + 2 = 5
3 + 1 + 0 = 4
3 + 1 + 1 = 5
3 + 2 + 0 = 5
4 + 0 + 0 = 4
4 + 0 + 1 = 5
4 + 1 + 0 = 5
```

Python Operators

- 1 + 1 = 2
- 2 – 3 = -1
- 2 * 3 = 6
- 2 / 3 = 0.667
- 2 ** 3 = 8

- 'hello' + 'world’ = ‘helloworld’
- 'hello’ + ’ ’ + 'world’ = ‘hello world’
- 'hello’ * 'world’ = Error
- 'hello’ / 'world’ = Error
- 'hello' * 3 = 'hellohellohello'

Python Operators

- [1] + [2] = [1, 2]
- [1] + 2 = Error
- [1] – [2] = Error
- [1] – [2] = Error
- [1, ‘hello’] + [4j, [1, 2]] = [1, 'hello', 4j, [1, 2]]
-

```
[20]  1  x = [1]
      2  x.append(2)
```

```
[21]  1  x
```

```
[1, 2]
```

```
[22]  1  x = [1]
      2  x.append([2])
```

```
[23]  1  x
```

```
[1, [2]]
```

```
[29]  1  x = []
      2  x.append('world!!')
      3  x.append([1, 2, 3])
      4  x.append(2)
```

```
[30]  1  x
```

```
['world!!', [1, 2, 3], 2]
```

Python Operators

```
[20]  1  x = [1]
      2  x.append(2)
```

```
[21]  1  x
```

```
[1, 2]
```

```
[22]  1  x = [1]
      2  x.append([2])
```

```
[23]  1  x
```

```
[1, [2]]
```

```
[29]  1  x = []
      2  x.append('world!!')
      3  x.append([1, 2, 3])
      4  x.append(2)
```

```
[30]  1  x
```

```
['world!!', [1, 2, 3], 2]
```

How to make a list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

- X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- X = list(range(1, 10))
- Using for loop
- List comprehension

Creating list

[6]
```
X = []  # create an empty list
X.append('world!!') # add new element to list using append method
X.append([1, 2, 3])
X.append(2)
```

[7]
```
X
```

```
['world!!', [1, 2, 3], 2]
```

[8]
```
X = []

for i in range(10):
    X.append(i)
```

[9]
```
X
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Exercise: create a list X, where
X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

[11]
```
X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Creating list

[6]
```python
X = []  # create an empty list
X.append('world!!') # add new element to list using append method
X.append([1, 2, 3])
X.append(2)
```

[7]
```python
X
```

```
['world!!', [1, 2, 3], 2]
```

[8]
```python
X = []

for i in range(10):
    X.append(i)
```

[9]
```python
X
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[10]
```python
X = []

for i in range(1, 11):
    X.append(i)
```

[11]
```python
X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Creating list

```
[8]  1  X = []
     2
     3  for i in range(10):
     4  |  X.append(i)
```

```
[9]  1  X
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[10]  1  X = []
      2
      3  for i in range(1, 11):
      4  |  X.append(i)
```

```
[11]  1  X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Exercises: create a list X,

1. X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2. X = [2, 4, 6, 8, 10, 12, 14, 16]
3. X = [1, -2, 3, -4, 5, -6, 7, -8, 9, -10]
4. X = [0, 1, 4, 9, 16, 25, 36, 49, 64]

Note:
range(*start*, *stop*, *step*)

List comprehension (set-builder form)

$\{n | n \in \mathbb{Z}^+\} = \{1, 2, 3, 4, \dots\}$

$\{x \in \mathbb{Z} | x > 3\} = \{4, 5, 6, \dots\}$

$\{n^2 | n \in \mathbb{Z}^+\} = \{1^2, 2^2, 3^2, \dots\}$

$\{2n + 1 | n \in \mathbb{Z}, 1 \leq n \leq 3\} = \{3, 5, 7\}$

$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] =$

$[n \mid \boxed{n \in [1,2,3,4,5,6,7,8,9,10]}]$

```
[24]  1  X = [n for n in range(1, 11)]
      2  X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
newlist = [expression for item in iterable]
```

List comprehension (set-builder form)

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] =$$

$$[n \mid \boxed{n \in [1,2,3,4,5,6,7,8,9,10]}]$$

```
[24]  1  X = [n for n in range(1, 11)]
      2  X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Exercises

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

List comprehension (set-builder form)

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] =$$

$$[n \mid \boxed{n \in [1,2,3,4,5,6,7,8,9,10]}]$$

```
[24]  1  X = [n for n in range(1, 11)]
      2  X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

=

```
1  X = []
2
3  for n in range(1, 11):
4      X.append(n)
```

```
1  X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Exercises

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

List comprehension (set-builder form)

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] =$$

$$[n \mid \boxed{n \in [1,2,3,4,5,6,7,8,9,10]}]$$

```
[24]  1  X = [n for n in range(1, 11)]
      2  X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

=

```
1  X = []
2
3  for n in range(1, 11):
4      X.append(n)
```

```
1  X
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Exercises `[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]`

Creating list (condition)

```
Fruits = ["apple", "banana", "cherry", "kiwi", "mango”, “Avocado”]
Text_a = []

for f in Fruits:
      if "a" in f:
            Text_a.append(f)

print(Text_a)
```

=> ["apple", "banana", "mango”, “Avocado”]

```
[89]  1  1 == 2
```

False

```
[90]  1  'test' == "test"
```

True

```
[91]  1  'b' in 'babna'
```

True

```
[92]  1  'b' in [1, 2, 3, 'babna']
```

False

```
[93]  1  1 == 1 and 1 == 2
```

False

```
1  1 == 1 or 1 == 2
```

True

Exercises: create a list (each word starts with a or A)

1. Text_a = [apple, Avocado]

Creating list (condition)

```
Fruits = ["apple", "banana", "cherry", "kiwi", "mango”, “Avocado”]
Text_a = []

for f in Fruits:
        if "a" in f:
                Text_a.append(f)

print(Text_a)
```

```
=> ["apple", "banana", "mango”, “Avocado”]
```

List Comprehension:

```
Text_a = [f for f in Fruits if "a" in f]
```

Exercises: create a list (each word starts with a or A)

1. Text_a = [apple, Avocado]

Loop with index

```
[ ]  1  Fruits = ["apple", "banana", "cherry", "kiwi", "mango", "Avocado"]
```

```
1  for fruit in Fruits:
2    print(f'fruit = {fruit}')
```

```
fruit = apple
fruit = banana
fruit = cherry
fruit = kiwi
fruit = mango
fruit = Avocado
```

```
[328]  1  for (i, fruit) in enumerate(Fruits):
       2    print(f'fruit {i} = {fruit}')
```

```
fruit 0 = apple
fruit 1 = banana
fruit 2 = cherry
fruit 3 = kiwi
fruit 4 = mango
fruit 5 = Avocado
```

Loop with index

```
[ ]  1  Fruits = ["apple", "banana", "cherry", "kiwi", "mango", "Avocado"]
```

```
1  for fruit in Fruits:
2  |  print(f'fruit = {fruit}')
```

```
fruit = apple
fruit = banana
fruit = cherry
fruit = kiwi
fruit = mango
fruit = Avocado
```

```
[328]  1  for (i, fruit) in enumerate(Fruits):
       2  |  print(f'fruit {i} = {fruit}')
```

```
fruit 0 = apple
fruit 1 = banana
fruit 2 = cherry
fruit 3 = kiwi
fruit 4 = mango
fruit 5 = Avocado
```

Exercise:

Hint: ‘hello’.upper() ==> ’HELLO’

```
fruit 1 = APPLE
fruit 2 = BANANA
fruit 3 = CHERRY
fruit 4 = KIWI
fruit 5 = MANGO
fruit 6 = AVOCADO
```

Loop with zip

```
[317]  1  Fruits = ["apple", "banana", "cherry", "kiwi", "mango", "Avocado"]
       2  Prices = [     45,       37,       60,     98,      35,       120]
```

```
[318]  1  for (fruit, price) in zip(Fruits, Prices):
       2  |  print(f'{fruit} = {price} baht')
```

```
apple = 45 baht
banana = 37 baht
cherry = 60 baht
kiwi = 98 baht
mango = 35 baht
Avocado = 120 baht
```

Exercise:

Hint: ‘hello’.ljust(7) ==> ’hello ’

Hint: print(f’n={120:4d}’) ==> ‘n= 120’

```
1. APPLE    =   45 baht
2. BANANA   =   37 baht
3. CHERRY   =   60 baht
4. KIWI     =   98 baht
5. MANGO    =   35 baht
6. AVOCADO  =  120 baht
```

```
[ ]  1  Fruits = ["apple", "banana", "cherry", "kiwi",
```

```
1  for fruit in Fruits:
2  |  print(f'fruit = {fruit}')
```

```
fruit = apple
fruit = banana
fruit = cherry
fruit = kiwi
fruit = mango
fruit = Avocado
```

```
[328]  1  for (i, fruit) in enumerate(Fruits):
       2  |  print(f'fruit {i} = {fruit}')
```

```
fruit 0 = apple
fruit 1 = banana
fruit 2 = cherry
fruit 3 = kiwi
fruit 4 = mango
fruit 5 = Avocado
```

Loop with zip

```
[317]  1  Fruits = ["apple", "banana", "cherry", "kiwi", "mango", "Avocado"]
       2  Prices = [     45,       37,       60,     98,      35,       120]
```

```
[318]  1  for (fruit, price) in zip(Fruits, Prices):
       2  |  print(f'{fruit} = {price} baht')
```

```
apple = 45 baht
banana = 37 baht
cherry = 60 baht
kiwi = 98 baht
mango = 35 baht
Avocado = 120 baht
```

```
[332]  1  for (i, (fruit, price)) in enumerate(zip(Fruits, Prices)):
       2  |  print(f'{i+1}. {fruit.upper().ljust(8)} = {price:4d} baht')
```

```
1. APPLE    =   45 baht
2. BANANA   =   37 baht
3. CHERRY   =   60 baht
4. KIWI     =   98 baht
5. MANGO    =   35 baht
6. AVOCADO  =  120 baht
```

Loop with index

```
[305]  1  Fruits = ["apple", "banana", "cherry", "kiwi", "mango", "Avocado"]
```

```
[306]  1  for fruit in Fruits:
       2  |  print(f'fruit = {fruit}')
```

```
fruit = apple
fruit = banana
fruit = cherry
fruit = kiwi
fruit = mango
fruit = Avocado
```

```
[307]  1  for i, fruit in enumerate(Fruits):
       2  |  print(f'fruit {i} = {fruit}')
```

```
fruit 0 = apple
fruit 1 = banana
fruit 2 = cherry
fruit 3 = kiwi
fruit 4 = mango
fruit 5 = Avocado
```

```
[309]  1  for i, fruit in enumerate(Fruits):
       2  |  print(f'fruit {i+1} = {fruit.upper()}')
```

```
fruit 1 = APPLE
fruit 2 = BANANA
fruit 3 = CHERRY
fruit 4 = KIWI
fruit 5 = MANGO
fruit 6 = AVOCADO
```

Outlines

- Data types (numeric)
  - Integer ex. -1
  - Real ex. 1.0
  - Complex ex. 1+2j

- Datatypes (sequence)
  - String ex. ‘hello’
  - List ex. [0, 1.1, 2, ‘x’, ‘abc’, 1-2j, [0, 1], (2, 1)]
  - Tuple ex. (0, 1.1, 2, ‘x’, ‘abc’, 1-2j, [0, 1], (2, 1))
  - Set ex. {0, 1.1, 2, ‘x’, ‘abc’, 1-2j, [0, 1], (2, 1)}
  - Dictionary ex. {1: ‘test’, “x”: 4, 1.2: 3.4, (0, 1): (2, 1)}

List – Tuple – Set

```
[119] 1  X = [1, 2, 4, 'X', 4]
```

```
[120] 1  X
```

```
[1, 2, 4, 'X', 4]
```

```
[121] 1  X[0] = 5
```

```
[122] 1  X
```

```
[5, 2, 4, 'X', 4]
```

```
[127] 1  Y = (1, 2, 4, 'X', 4)
```

```
[128] 1  Y[0]
```

```
1
```

```
[129] 1  Y[0] = 5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most r
<ipython-input-129-6de98b3a371a> in <cell line: 1>()
----> 1 Y[0] = 5

TypeError: 'tuple' object does not support item assignment
```

```
[196] 1  Y = (i/2 for i in range(1, 11))
```

```
[197] 1  Y
```

```
<generator object <genexpr> at 0x7b8964b4e3b0>
```

```
[198] 1  for y in Y:
      2      print(y)
```

```
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
```

```
[130] 1  Z = {1, 2, 4, 'X', 4}
```

```
[131] 1  Z
```

```
{1, 2, 4, 'X'}
```

```
[132] 1  Z[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Trac
<ipython-input-132-8d182828ef3f> in <cell line
----> 1 Z[0]

TypeError: 'set' object is not subscriptable
```

Attribute

Attributes in Python are variables associated with an object and are used to store data related to the object.

```
[168] 1  x = 1
```

```
[169] 1  dir(x)
```

```
['__format__',
 '__ge__',
 '__getattribute__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__index__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__invert__',
 '__le__',
 '__lshift__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
 '__or__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__rand__',
 '__rdivmod__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rfloordiv__',
 '__rlshift__',
 '__rmod__',
 '__rmul__',
 '__ror__',
 '__round__',
 '__rpow__',
 '__rrshift__',
 '__rshift__',
 '__rsub__',
 '__rtruediv__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 '__trunc__',
 '__xor__',
 'as_integer_ratio',
 'bit_count',
 'bit_length',
 'conjugate',
 'denominator',
 'from_bytes',
 'imag',
 'numerator',
 'real',
 'to_bytes']
```

```
[168] 1  x = 1
```

```
[170] 1  dir(int)
```

```
'__format__',
'__ge__',
'__getattribute__',
'__getnewargs__',
'__gt__',
'__hash__',
'__index__',
'__init__',
'__init_subclass__',
'__int__',
'__invert__',
'__le__',
'__lshift__',
'__lt__',
'__mod__',
'__mul__',
'__ne__',
'__neg__',
'__new__',
'__or__',
'__pos__',
'__pow__',
'__radd__',
'__rand__',
'__rdivmod__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rfloordiv__',
'__rlshift__',
'__rmod__',
'__rmul__',
'__ror__',
'__round__',
'__rpow__',
'__rrshift__',
'__rshift__',
'__rsub__',
'__rtruediv__',
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__truediv__',
'__trunc__',
'__xor__',
'as_integer_ratio',
'bit_count',
'bit_length',
'conjugate',
'denominator',
'from_bytes',
'imag',
'numerator',
'real',
'to_bytes']
```

```
1  x = -11
```

```
[192] 1  x.real
```

```
-11
```

```
[193] 1  x.imag
```

```
0
```

```
[194] 1  x.bit_length
```

```
<function int.bit_length()>
```

```
[195] 1  x.bit_length()
```

```
4
```

Exercises: filter out items that start with "_":

```
Hint: result = [? for ?? in dir(int) if ???]


or


result = []
for ?? in dir(int):
    if ???:
        result.append(?)
```

```
[203] 1  [string for string in dir(str) if string[0] != '_']
```

```
['capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'maketrans',
 'partition',
 'removeprefix',
 'removesuffix',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
 'splitlines',
 'startswith',
 'strip',
 'swapcase',
 'title',
 'translate',
 'upper',
 'zfill']
```

Examples:

```
[220] 1  "Python's list comprehensions simplified filtering."
```

```
'Python's list comprehensions simplified filtering.'
```

```
[221] 1  "Python's list comprehensions simplified filtering.".split()
```

```
["Python's", 'list', 'comprehensions', 'simplified', 'filtering.']
```

```
[222] 1  "Python's list comprehensions simplified filtering.".upper()
```

```
'PYTHON'S LIST COMPREHENSIONS SIMPLIFIED FILTERING.'
```

```
[223] 1  "Python's list comprehensions simplified filtering.".startswith('Py')
```

```
True
```

```
[229] 1  X = [2**2 for i in range(1, 11)]
```

```
[230] 1  X
```

```
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
```

```
[231] 1  [att for att in dir(X) if att[0] != '_']
```

```
['append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

```
[232] 1  X.append(10)
      2  X
```

```
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 10]
```

```
[234] 1  X.insert
```

```
<function list.insert(index, object, /)>
```

```
[235] 1  X.insert(1, 5)
```

```
[236] 1  X
```

```
[4, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 10]
```

Exercises: create list:

1.[1, 1, 1, 1, 1, 5, 1, 1]

2.[2, 4, 6, 8, …, 36, 99]

3.[1, 2, 3, …, 25, ‘python’, 26, …, 32, 33, 34, …, 40, ‘python’, 41, …, 50]

NumPy

Numerical computing tools in Python

```
[203]  1  import numpy as np
```

```
[204]  1  X = [1, 2, 3, 'hello']
```

```
[205]  1  X = np.array(X)
       2  X
```

```
array(['1', '2', '3', 'hello'], dtype='<U21')
```

```
[206]  1  Y = [1, 2, 3]
```

```
[207]  1  Y = np.array(Y)
       2  Y
```

```
array([1, 2, 3])
```

```
[208]  1  [att for att in dir(Y) if att[0] != '_']
```

```
 'conj',
 'conjugate',
 'copy',
 'ctypes',
 'cumprod',
 'cumsum',
 'data',
 'diagonal',
 'dot',
 'dtype',
```

[203]
```python
import numpy as np
```

[204]
```python
X = [1, 2, 3, 'hello']
```

[205]
```python
X = np.array(X)
X
```

```
array(['1', '2', '3', 'hello'], dtype='<U21')
```

[206]
```python
Y = [1, 2, 3]
```

[207]
```python
Y = np.array(Y)
Y
```

```
array([1, 2, 3])
```

[208]
```python
[att for att in dir(Y) if att[0] != '_']
```

```
 'conj',
 'conjugate',
 'copy',
 'ctypes',
 'cumprod',
 'cumsum',
 'data',
 'diagonal',
 'dot',
 'dtype',
```

[ ]
```python
Y.std()
```

```
0.816496580927726
```

[ ]
```python
Y.sum()
```

```
6
```

[209]
```python
Y.cumsum()
```

```
array([1, 3, 6])
```

[210]
```python
Y.tolist()
```

```
[1, 2, 3]
```

[211]
```python
Y.shape
```

```
(3,)
```

Array = n dimensional List

```
B = np.array([[2, 9, 3], [4, 76, 23], [2, 1, 1]])
B
```

```
array([[ 2,  9,  3],
       [ 4, 76, 23],
       [ 2,  1,  1]])
```

```
[260]  B ** 2
```

```
array([[   4,   81,    9],
       [  16, 5776,  529],
       [   4,    1,    1]])
```

Exercises: create an array: $\begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix}$

Array = List (n dimensional list)

```
[268]  1  A = [[2, 9, 3], [4, 76, 23], [2, 1, 1]]
       2  A
```

```
[[2, 9, 3], [4, 76, 23], [2, 1, 1]]
```

```
[269]  1  A[1][2]
```

```
23
```

```
[270]  1  B = np.array([[2, 9, 3], [4, 76, 23], [2, 1, 1]])
       2  B
```

```
array([[ 2,  9,  3],
       [ 4, 76, 23],
       [ 2,  1,  1]])
```

```
[271]  1  B[1, 2]
```

```
23
```

Matrix = array (2-dimensional array)

```
[262]  C = np.matrix([[2, 1, 3], [3, 6, 7], [5, 8, 7]])
       C
```

```
matrix([[2, 1, 3],
        [3, 6, 7],
        [5, 8, 7]])
```

```
[263]  C ** 2
```

```
matrix([[ 22,  32,  34],
        [ 59,  95, 100],
        [ 69, 109, 120]])
```

```
[301]  C = np.matrix("[2 1 3; 3 6 7; 5 8 7]")
       C
```

```
matrix([[2, 1, 3],
        [3, 6, 7],
        [5, 8, 7]])
```

Dictionary

- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*, which can be any immutable type; strings and numbers can always be keys.
- Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.
- You can't use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like **append**() and **extend**().

```python
# creating a dictionary
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```

[10]

```python
# creating a dictionary
country_capitals = {"Germany": "Berlin", "Canada": "Ottawa", "England": "London"}

# printing the dictionary
print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```

[12]

```python
country_capitals['Germany']
```

```
'Berlin'
```

[14]

```python
country_capitals['Thailand'] = 'Bangkok'
print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London', 'Thailand': 'Bangkok'}
```

```
[15]  1  dir(country_capitals)
```

```
['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__ror__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

```
1  # creating a dictionary
2  country_capitals = {
3    "Germany": "Berlin",
4    "Canada": "Ottawa",
5    "England": "London"
6  }
7
8  # printing the dictionary
9  print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```

```
[10]  1  # creating a dictionary
      2  country_capitals = {"Germany": "Berlin", "Canada": "Ottawa", "England": "London"}
      3
      4  # printing the dictionary
      5  print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```

```
[12]  1  country_capitals['Germany']
```

```
'Berlin'
```

```
[14]  1  country_capitals['Thailand'] = 'Bangkok'
      2  print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London', 'Thailand': 'Bangkok'}
```

```
[18]  1  country_capitals.items()
```

```
dict_items([('Germany', 'Berlin'), ('Canada', 'Ottawa'), ('England', 'London'), ('Thailand', 'Bangkok')])
```

```
[20]  1  country_capitals.values()
```

```
dict_values(['Berlin', 'Ottawa', 'London', 'Bangkok'])
```

```
[21]  1  country_capitals.keys()
```

```
dict_keys(['Germany', 'Canada', 'England', 'Thailand'])
```

```
[22]  1  country_capitals.pop('Thailand')
      2  country_capitals
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```

```
[24]  1  import pandas as pd
```

```
[42]  1  data_dict = {
      2      'column_1': [1, 2, 3, 4, 5, 6],
      3      'column_2': [1, 2, 3, 4, 5, 6],
      4      'column_3': [1, 2, 3, 4, 5, 6],
      5  }
      6  pd.DataFrame(data_dict)
```

	column_1	column_2	column_3
0	1	1	1
1	2	2	2
2	3	3	3
3	4	4	4
4	5	5	5
5	6	6	6

```
[39]  1  pd.DataFrame({
      2      'col1': ['row1', 'row2'],
      3      'col2': ['row1', 'row2'],
      4      'col3': ['row1', 'row2'],
      5  })
```

	col1	col2	col3
0	row1	row1	row1
1	row2	row2	row2

List of loading functions to dataframe (import pandas as pd)

- `df = pd.read_csv(…)  => dir or url`
- `df = pd.read_excel(…)  => dir or url`

List of important pandas functions

- `df.info()`
- `df.head(…)`
- `df.tail(…)`
- `df.dropna()`
- `df.describe()`
- `df.plot(…)`
- `df.groupby(…)`

https://cmu.to/py-link

Untitled20.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Files

..

sample_data

20240304130128_11474.xls

+ Code + Text

```
[2] import pandas as pd
```

```
[7] url = "https://www.nso.go.th/nsoweb/downloadFile/stat_main_nso/AkMX/file_th"
    pd.read_excel(url)
```

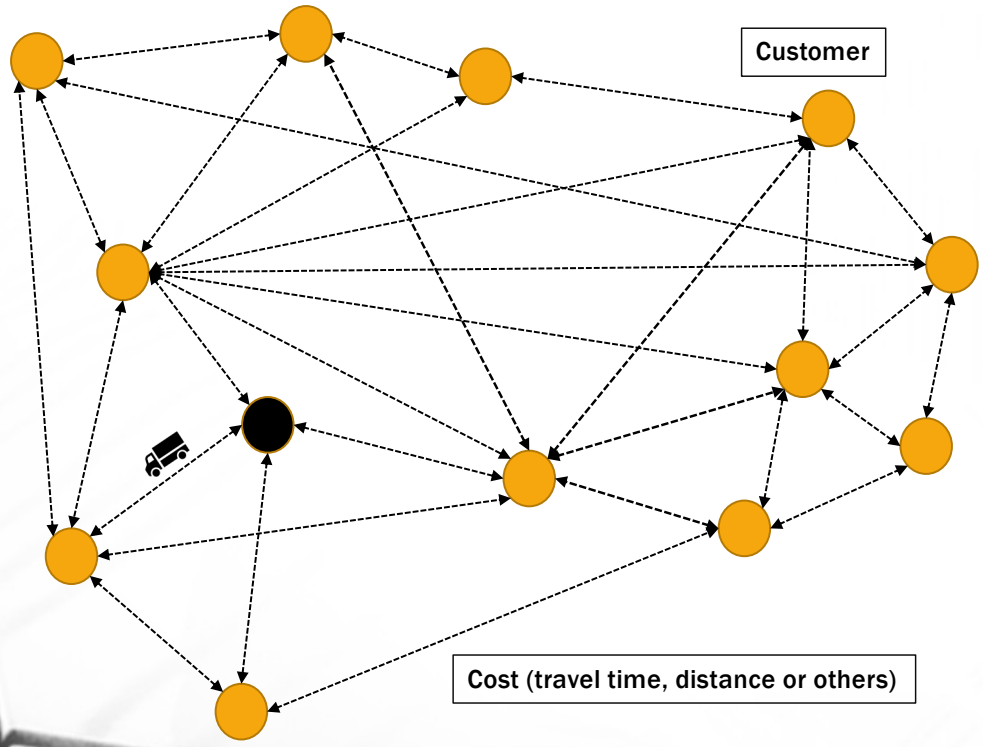
```
pd.read_excel('/content/20240304130128_11474.xls')
```

	ตาราง 15.1 รถจดทะเบียน (สะสม) ตามพระราชบัญญัติรถยนต์ พ.ศ. 2522 จำแนกตามประเภทรถ พ.ศ. 2557 – 2561	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6
0	Table 15.1 Vehicle Registered Under Motor Vehi...	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	ประเภทรถ	2557	2558	2559	2560	2561	Type of vehicle
3	NaN	(2014)	(2015)	(2016)	(2017)	(2018)	NaN
4	รวมยอด	1250896	1328670	1339664	1382389	1431443	Total
5	รถยนต์นั่งส่วนบุคคลไม่เกิน 7 คน	260383	277387	295475	315670	339014	Sedan (not more than 7 passengers)
6	รถยนต์นั่งส่วนบุคคลเกิน 7 คน	12585	12218	12123	12036	11900	Microbus & passenger van
7	รถยนต์บรรทุกส่วนบุคคล	126024	255432	233437	241980	249919	Van & pick up
8	รถยนต์สามล้อส่วนบุคคล	54	61	61	58	49	Motortricycle
9	รถยนต์รับจ้างระหว่างจังหวัด	-	-	-	-	-	Interprovincial taxi
10	รถยนต์รับจ้างบรรทุกคนโดยสารไม่เกิน 7 คน	350	316	404	387	345	Urban taxi
11	รถยนต์สี่ล้อเล็กรับจ้าง	28	27	27	28	-	Fixed route taxi
12	รถยนต์รับจ้างสามล้อ	1142	1141	1142	1149	1171	Motortricycle taxi (tuk tuk)
13	รถยนต์บริการธุรกิจ	54	155	200	208	204	Hotel taxi
14	รถยนต์บริการทัศนาจร	-	-	-	6	21	Tour taxi
15	รถยนต์บริการให้เช่า	-	-	-	-	-	Car for hire
16	รถจักรยานยนต์	755667	776769	791405	805350	823233	Motorcycle
17	รถแทรกเตอร์	3855	4193	4416	4424	4621	Tractor
18	รถบดถนน	312	349	388	430	436	Road roller
19	รถใช้งานเกษตรกรรม	28	29	30	34	34	Farm vehicle
20	รถพ่วง	17	19	22	41	38	Automobile trailer
21	รถจักรยานยนต์สาธารณะ	397	573	534	488	458	Public motorcycle
22	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23	ที่มา:	สำนักงานขนส่งจังหวัดเชียงใหม่	NaN	NaN	NaN	NaN	NaN
24	Source:	Chiang Mai Provincial Transport Office	NaN	NaN	NaN	NaN	NaN

```
[ ]
```

Disk 81.40 GB available

Traveling Salesmen Problem (TSP)



Given:

- The cost between each pair of customers

Constraints:

- each customer is visited by only one vehicle
- The salesman starts and ends at the same node

Objective:

- Minimize the cost of routing
  - Total distance
  - Total travel time

Vehicle Routing formulation

$$\text{minimize} \quad \sum_{k}\sum_{j}\sum_{i} c_{ij} x_{ij}^{k}$$

subject to

$$\sum_{j\in V} x_{0j}^{k} = 1, \qquad \forall k \in K = set\ of\ all\ vehicles$$

$$\sum_{j\in V} x_{j0}^{k} = 1, \qquad \forall k \in K$$

$$\sum_{k\in K}\sum_{i\in V} x_{ij}^{k} = 1, \qquad \forall j \in V = set\ of\ all\ customer\ node$$

$$\sum_{k\in K}\sum_{j\in V} x_{ij}^{k} = 1, \qquad \forall i \in V$$

$$\sum_{i\in V} x_{ij}^{k} - \sum_{l\in V} x_{jl}^{k} = 0, \qquad \forall \mathrm{j} \in V, k \in K$$

$$\sum_{i,j\in V, i\neq j} d_j x_{ij}^{k} \leq Q, \qquad k \in K$$

$$\sum_{i\in S, j\in S, i\neq j} x_{ij}^{k} = |S| - 1, \forall S \subset \{1,2,\dots,n-1\}, 2 \leq |S|$$

$x_{ij}^{k} = 1$ if vehicle k travels from customer i to customer j, 0 otherwise
c_{ij} = cost (distance or travel time) from i to j

(1) Each vehicle must leave from the depot

(2) Each vehicle must return to the depot

(3) Only one vehicle leaves a customer node

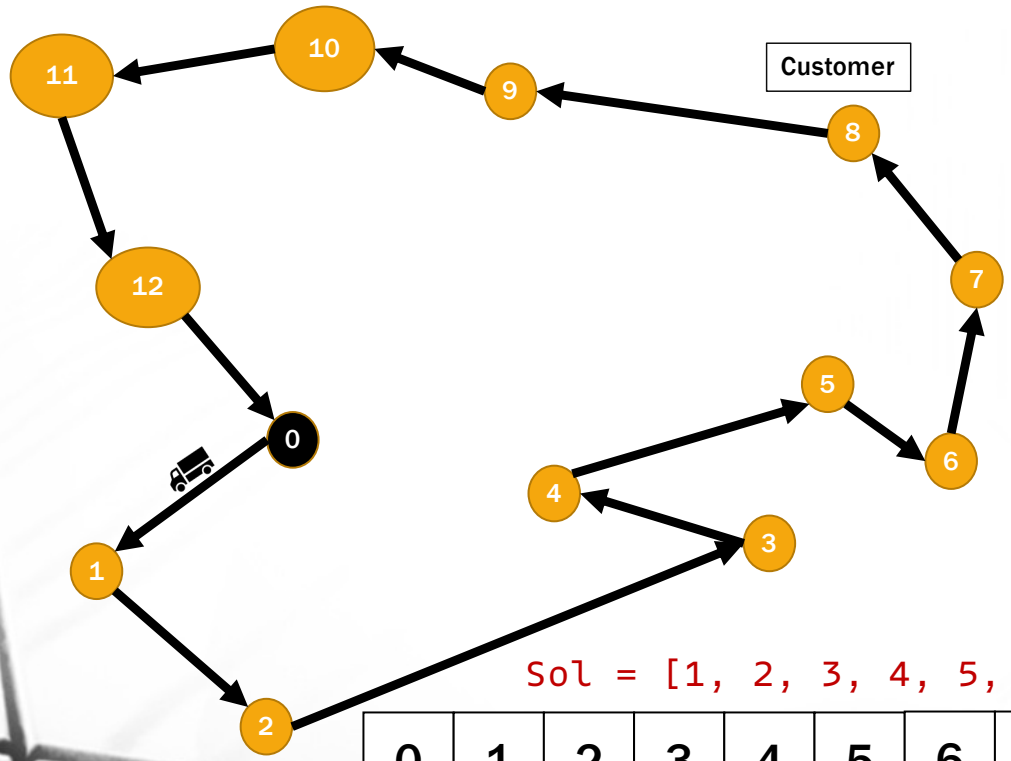
(4) Only one vehicle enters a customer node

(5) Vehicle that visits and leaves a customer node is the same vehicle

(6) Total demand not exceed maximum capacity of vehicle

(7) Subtour elimination constraints

Traveling Salesmen Problem (TSP)



$$\texttt{Total solutions} = 12! = 479,001,600$$

Number of customers	Total possible solutions	
10	3,628,800	=> 3 million (10^6)
13	6,227,020,800	=> 6 billion (10^9)
15	1,307,674,368,000	=> 1 trillion (10^{12})
20	2,432,902,008,176,640,000	=> (10^{18})
21	10^{19}	
22	10^{21}	

Frontier: 10^{18} **FLOPS**

Sol = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

0	1	2	3	4	5	6	7	8	9	10	11	12	0

Algorithm

- **Greedy algorithm**:

  - Insert the lowest distance

Iteration 1:

0	1	0

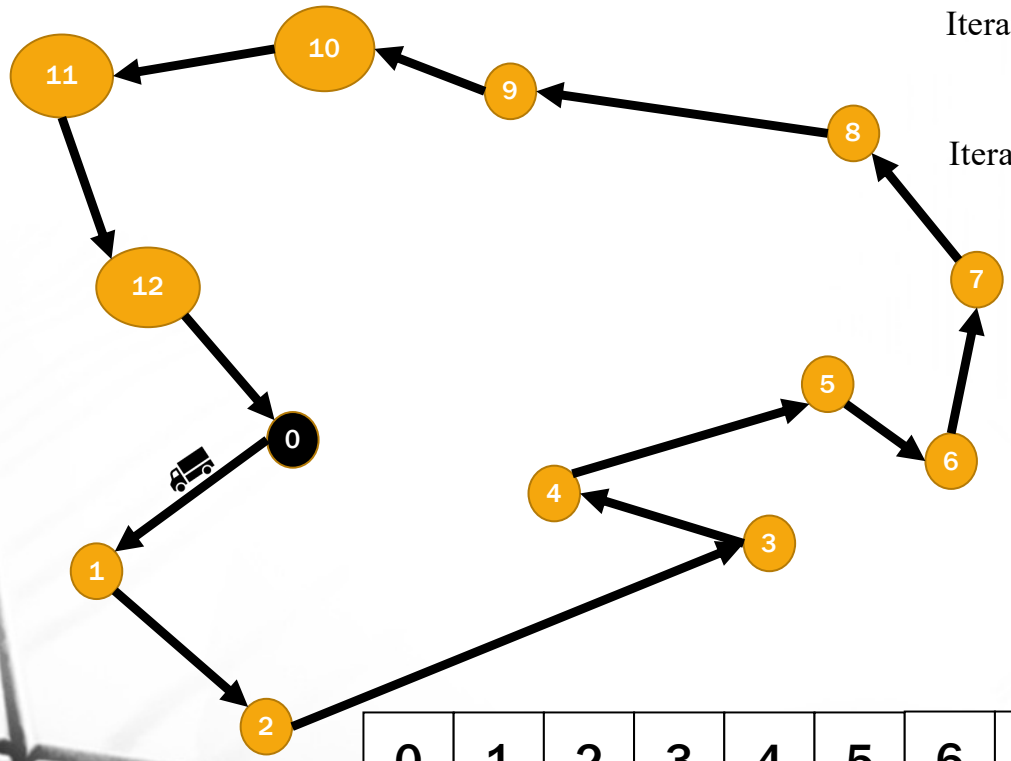
Iteration 2:

0	1	2	0

Iteration 3:

0	1	2	3	0

.
.
.



0	1	2	3	4	5	6	7	8	9	10	11	12	0