고려대학교 의료정보학과

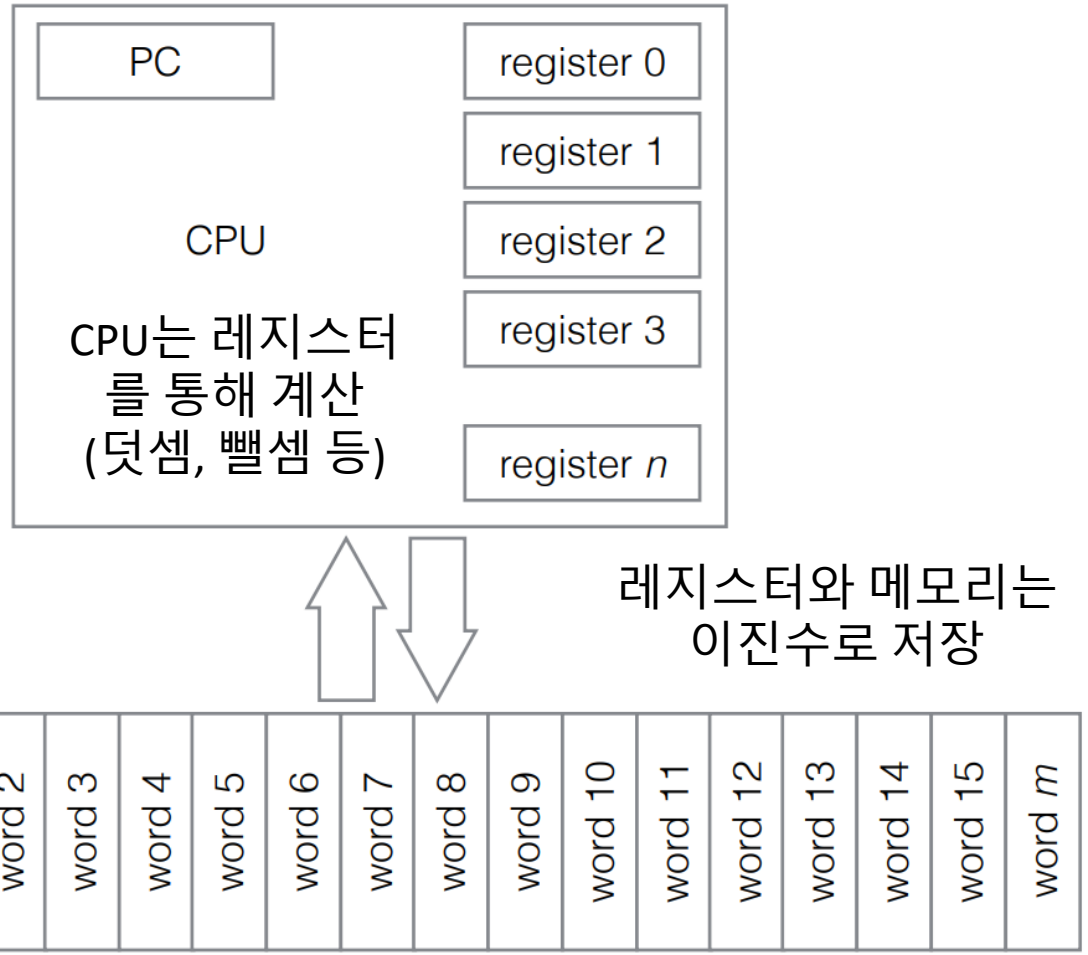
KU MEDICINE IMDL Intelligent Medical Data Lab

의료인공지능
파이썬 - 자료형과 제어문

고려대학교 의료빅데이터연구소
채민수(minsuchae@korea.ac.kr)

1. 자료형

- 자료형의 필요성
  - 정수의 덧셈, 뺄셈
    - add
    - sub
  - 실수의 덧셈, 뺄셈
    - fadd
    - fsub



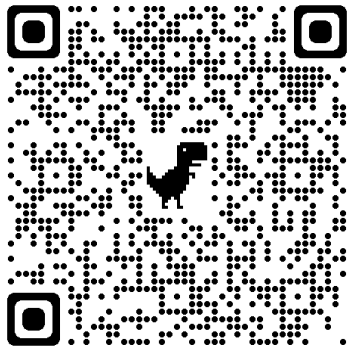
Reference : https://www.cs.cmu.edu/~ckingsf/class/02201-f15/lects/lec18-arch.pdf

1. 자료형

◦ 파이썬의 자료형 타입

- 숫자 타입

➢ int(-2147483648 ~ 2147483647)



```
print(1-2)                          [실행결과]
print(1+2)                          -1
print(-2147483648-1)                3
print(type(-2147483648-1))          -2147483649
print(2147483647+1)                 int
print(type(2147483647+1))           2147483648
                                    int
```

➢ float(부호비트:1비트, 지수부:11비트, 가수부:53비트)

➢ Decimal

➢ Fraction

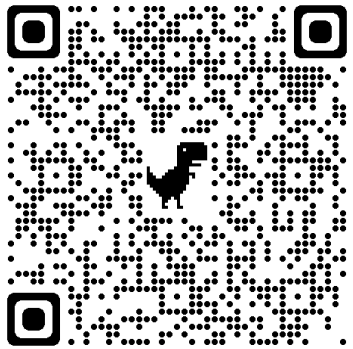
➢ Complex

1. 자료형

◦ 파이썬의 자료형 타입

- 문자 타입

➢ str



```
print('a')
print(type('a'))
print("alphabet")
print(type("alphabet"))
string = '''line1
line2
line3'''
print(string)
print(type(string))
string="""line1
line2
line3"""
print(string)
print(type(string))
```

[실행결과]

```
a
str
alphabet
str


line1₩nline2₩nline3
str


line1₩nline2₩nline3
str
```

1. 자료형

- 파이썬의 자료형 타입
  - 문자 타입
    - 이스케이프 문자(Escape character)
      - ₩₩ : backslash
      - ₩r : carriage return
      - ₩n : line feed
      - ₩t : tab
      - ₩' : single quotes
      - ₩" : double quotes
      - etc : ₩a, ₩b

1. 자료형

- 파이썬의 자료형 타입
  - 다중 라인 활용 - 주석

```
"""Wrapper allowing a stack of RNN cells to behave as a single cell.

Used to implement efficient stacked RNNs.

Args:
  cells: List of RNN cell instances.

Examples:

```python
batch_size = 3
sentence_max_length = 5
n_features = 2
new_shape = (batch_size, sentence_max_length, n_features)
x = tf.constant(np.reshape(np.arange(30), new_shape), dtype = tf.float32)

rnn_cells = [tf.keras.layers.LSTMCell(128) for _ in range(2)]
stacked_lstm = tf.keras.layers.StackedRNNCells(rnn_cells)
lstm_layer = tf.keras.layers.RNN(stacked_lstm)

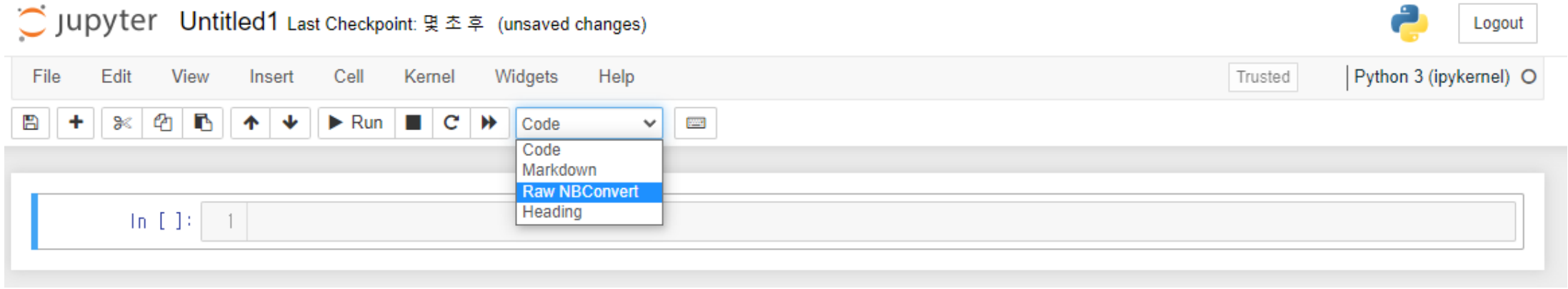
result = lstm_layer(x)
```
"""
```

Reference : https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/keras/layers/recurrent.py#L56

1. 자료형

◦ 파이썬의 자료형 타입

- 다중 라인 활용



1. 자료형

◦ 파이썬의 자료형 타입

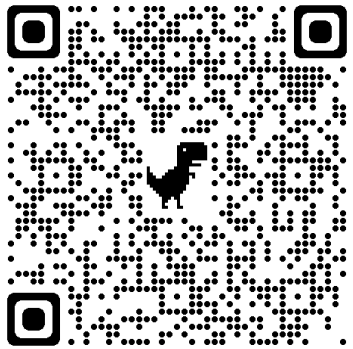
- 바이트 타입
  - ➢ bytes
  - ➢ bytearray
- 불리언 타입
  - ➢ bool

1. 자료형

◦ 파이썬의 자료형 타입

- 날짜 타입

➢ datetime



```
import datetime

date = datetime.datetime.strptime('2022-09-13', "%Y-%m-%d")
print(date)
print(type(date))
```

[실행결과]
2022-09-13 00:00:00
<class 'datetime.datetime'>

➢ https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes

1. 자료형

◦ 변수명 명명 규칙

- 예약어 사용 금지(if, else, elif, and, or, not, while, class, break, continue, yield, import, def).
- 한국어를 포함한 알파벳과, 언더바(_)로 변수명이 시작되어야 함
- 내장 함수명 사용 금지 권고
- 변경가능한 변수는 소문자로, 상수는 대문자로 권고

2. 자료 구조

◦ 배열 타입

- 순차적으로 데이터를 표현하는 자료형

- 다양한 자료형을 허용함

- list : 수정가능

- tuple : 수정불가(읽기 전용 데이터)

2. 자료 구조

- 배열 타입

Actual Address of the 1st element of the array is known as **Base Address (B)** Here it is 1100

Memory space acquired by every element in the Array is called **Width (W)** Here it is 4 bytes

Actual Address in the Memory	1100	1104	1108	1112	1116	1120
Elements	**15**	**7**	**11**	**44**	**93**	**20**
Address with respect to the Array (Subscript)	0	1	2	3	4	5

Lower Limit/Bound of Subscript **(LB)**

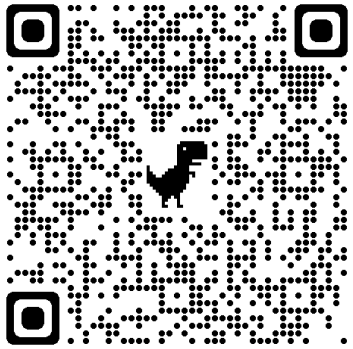
Reference : https://www.quora.com/What-is-the-difference-between-index-and-address-of-an-array-in-data-structures

2. 자료 구조

- List
  - [] 기호나 list함수를 통해 정의

```
a = [1,2,3,[4,5,6]]
print(a)
```

[실행결과]
[1, 2, 3, [4, 5, 6]]



  - List 내에 List를 넣음으로써 중첩시켜 2차원 이상을 표현
  - append : List의 마지막에 값 추가
  - insert : List의 특정 index에 원하는 값 추가
  - remove : List의 특정한 값을 삭제(인덱스 상 먼저 앞에 있는 값만 삭제)
  - https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types

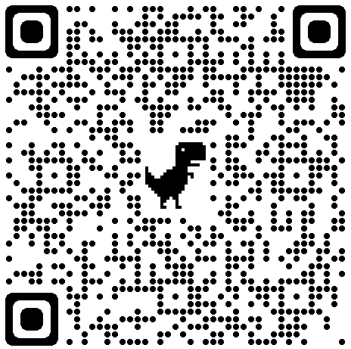
2. 자료 구조

◦ List 실습

환자 ID	키(cm)	몸무게(Kg)	SBP	DBP	맥박
1	170	100	130	120	100
2	160	80	110	80	105
3	150	40	100	70	80

```
data = [
    [1, 170, 100, 130, 120, 100],
    [2, 160, 80, 110, 100, 105],
    [3, 150, 40, 100, 70, 80]
]
print(data)
```

[실행결과]
[[1, 170, 100, 130, 120, 100], [2, 160, 80, 110, 100, 105], [3, 150, 40, 100, 70, 80]]



2. 자료 구조

- Tuple
  - () 기호나 두 개이상의 값을 표현하여 정의
  - 파이썬에서 두 개이상의 값을 하나의 값으로 표현
  - 값 변경 불가
  - https://docs.python.org/3/library/stdtypes.html#immutable-sequence-types

2. 자료 구조

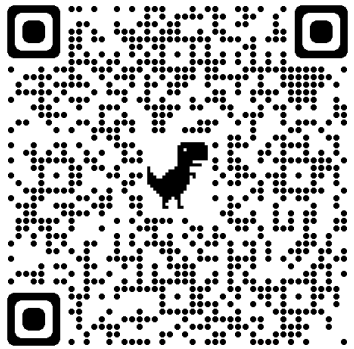
◦ Tuple 실습

```
x = 10
y = 5
tmp = x
x = y
y = tmp
print(x, y)
```

[실행결과]
5 10

```
x = 10
y = 5
x, y = y, x
print(x, y)
```

[실행결과]
5 10



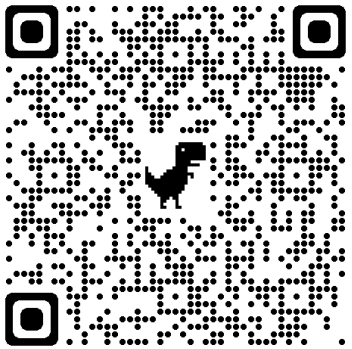
2. 자료 구조

◦ Set

- 집합으로 중복된 값을 포함하지 않고 저장함

```
my_list = [1,1,2,3,4,5,6,7,8]
my_set = set([1,1,2,3,4,5,6,7,8])
print(my_list)
print(my_set)
```

[실행결과]
[1, 1, 2, 3, 4, 5, 6, 7, 8]
{1, 2, 3, 4, 5, 6, 7, 8}



- Dictionary 의 key가 set으로 구성되어 있음

2. 자료 구조

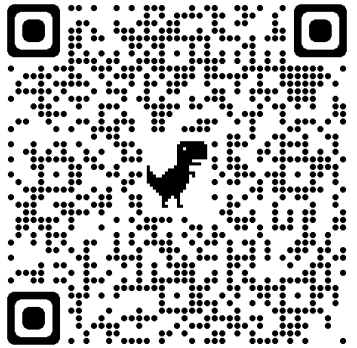
- Dictionaries
  - 배열은 정수 인덱스를 통해 접근하나, 문자열 등을 통해 접근
  - 키를 접근하여 할당 시 덮어씌움(overwrite)

```
dictionary = {'홍길동':'홍길동@customdomain.net', '아무개':'아무개@customdomain.net'}
print(dictionary['홍길동'])
```

```
[실행결과]
홍길동@customdomain.net
```

```
print(dictionary['채민수'])
```

```
[실행결과]
KeyError: '채민수'
```



3. 연산자

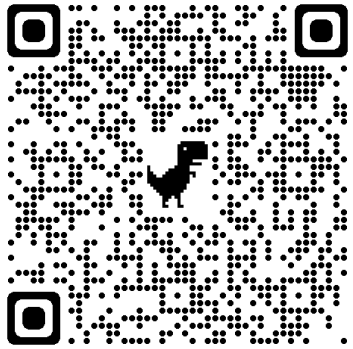
- 산술 연산자
  - 할당 =
  - 덧셈 +
  - 뺄셈 -
  - 곱셈 *
  - 제곱 **
  - 나눗셈 /, //
  - 나머지연산 %

3. 연산자

- 산술 연산자 실습

```
a = 5
b = 3
print(a+b)
print(a-b)
```

[실행결과]
8
2



```
print(a * b)
```

[실행결과]
15

```
print(a ** b)
```

[실행결과]
125

3. 연산자

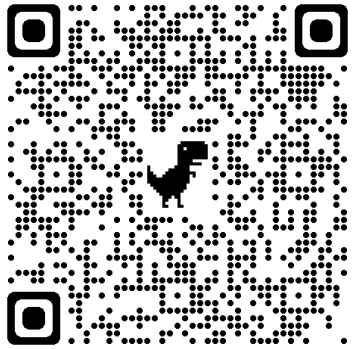
- 산술 연산자 실습

```
a = 5
b = '3'
print(a+b)
```

[실행결과]
TypeError: unsupported operand type(s) for +: 'int' and 'str'

```
print(a+int(b))
```

[실행결과]
8



3. 연산자

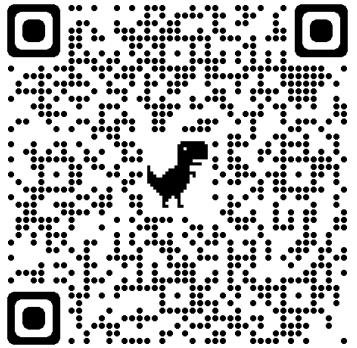
◦ 복합 산술 연산자

- 산술 연산 후 대입 수행

- e.g.; +=, -=, *=, /=, //=, %=, **=

```
result = 0
result += 2
print(result)
```

[실행결과]
2



3. 연산자

- 비교 연산자
  - < : 미만
  - <= : 이하
  - == : 같음
  - != : 같지 않음
  - > : 초과
  - >= : 이상

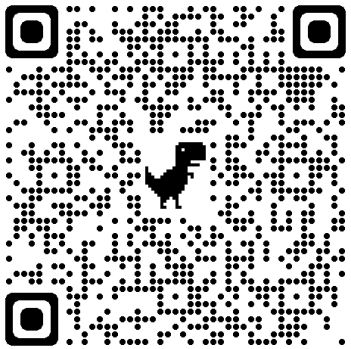
3. 연산자

◦ 비교 연산자 실습

```
import datetime

today = datetime.datetime.strptime('2022-09-13', "%Y-%m-%d")
tomorrow = datetime.datetime.strptime('2022-09-14', "%Y-%m-%d")
print(today < tomorrow)
```

[실행결과]
True



```
import datetime

yesterday = datetime.datetime.strptime('2022-09-12', "%Y-%m-%d")
today = datetime.datetime.strptime('2022-09-13', "%Y-%m-%d")
tomorrow = datetime.datetime.strptime('2022-09-14', "%Y-%m-%d")
print(yesterday < today < tomorrow)
```

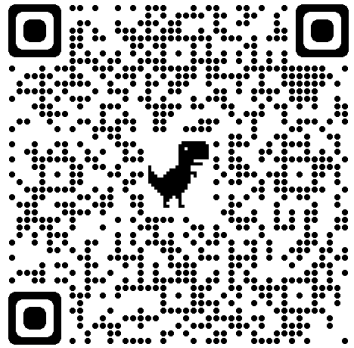
[실행결과]
True

3. 연산자

- 식별 연산자
  - is : 같은 메모리를 가리키고 있으면 참
  - is not : 다른 메모리를 가리키고 있으면 참

```
a = 10
b = 10
c = 10.0
print(a == b)
print(a is b)
print(a == c)
print(a is c)
```

[실행결과]
True
True
True
False



3. 연산자

- 멤버 연산자
  - in : 해당 값을 포함하고 있으면 참
  - not in : 다른 메모리를 가리키고 있으면 참

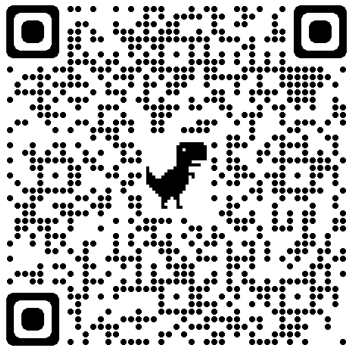
- 논리 연산자
  - and : 두 조건식이 모두 참일 경우 참
  - or : 두 조건식 중 하나라도 참일 경우 참
  - not : 논리 부정으로 참은 거짓, 거짓은 참으로 변경

3. 연산자

- 멤버 연산자

```
dictionary = {'홍길동':'홍길동@customdomain.net', '아무개':'아무개@customdomain.net'}
print('홍길동' in dictionary)
print('채민수' in dictionary)
```

[실행결과]
True
False



3. 연산자

- 비트연산자
  - & : 비트 단위로 논리곱 연산 수행
  - | : 비트 단위로 논리합 연산 수행
  - ^ : 비트 단위로 배타적 논리합(exclusive or) 연산 수행
  - ~ : 비트 단위로 논리 부정 연산 수행
  - << : 비트 단위로 왼쪽으로 시프트
  - >> : 비트 단위로 오른쪽으로 시프트

3. 연산자

◦ 연산자 우선 순위

우선순위	연산자	설명
1	(,)	소괄호
2	**	제곱
3	+x, -x, ~x	부호있는 변수값, 비트단위 논리 부정
4	*, /, //, %	곱셈, 나눗셈, 나머지 연산
5	+, -	덧셈, 뺄셈
6	<<, >>	비트 단위 시프트
7	&	비트 단위 논리곱
8	^	비트단위 배타적 논리합
9	|	비트단위 논리합
10	==, !=, >, >=, <, <=, is, is not, in, not in	비교 연산자, 멤버 연산자, 식별 연산자
11	not	논리 부정
12	and	논리 곱
13	or	논리 합

4. 조건문

◦ if 조건문

- 평가식을 통해 특정 실행문으로 분기

- if, elif, else 키워드를 통해 비교함

4. 조건문

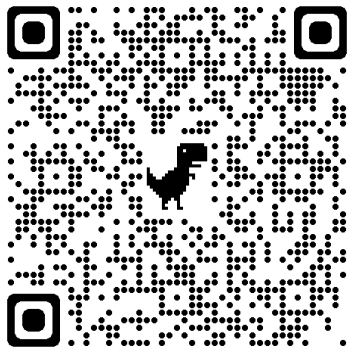
◦ if 조건문 실습

```
data = [
    [1, 170, 100, 130, 120, 100],
    [1, 170, 100, 130, 120, 103],
    [1, 170, 100, 130, 120, 105],
    [2, None, None, 110, 100, 105],
    [2, None, None, 110, 100, 102],
    [2, 160, 80, 110, 100, 103],
    [3, None, None, 100, 70, 80],
    [3, None, None, 100, 70, 85],
    [3, None, None, 100, 70, 90]
]
print(data)
```

[실행결과]
[[1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 103], [1, 170, 100, 130, 120, 105], [2, None, None, 110, 100, 105], [2, None, None, 110, 100, 102], [2, 160, 80, 110, 100, 103], [3, None, None, 100, 70, 80], [3, None, None, 100, 70, 85], [3, None, None, 100, 70, 90]]

4. 조건문

◦ if 조건문 실습



```
my_dictionary = {}
my_dictionary[1] = {'height':170, 'weight':100}
my_dictionary[2] = {'height':160, 'weight':80}
my_dictionary[3] = {}

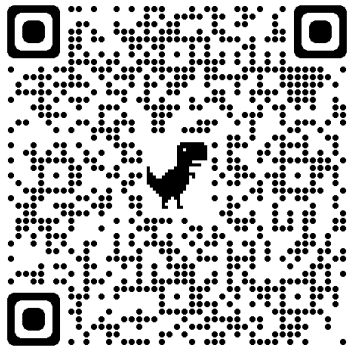
data[3][1] = my_dictionary[2]['height']
data[3][2] = my_dictionary[2]['weight']
data[4][1] = my_dictionary[2]['height']
data[4][2] = my_dictionary[2]['weight']

print(data)
```

[실행결과]
[[1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 103], [1, 170, 100, 130, 120, 105], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 102], [2, 160, 80, 110, 100, 103], [3, None, None, 100, 70, 80], [3, None, None, 100, 70, 85], [3, None, None, 100, 70, 90]]

4. 조건문

◦ if 조건문 실습



```
data[5][1] = my_dictionary[3]['height']
data[5][2] = my_dictionary[3]['weight']
```

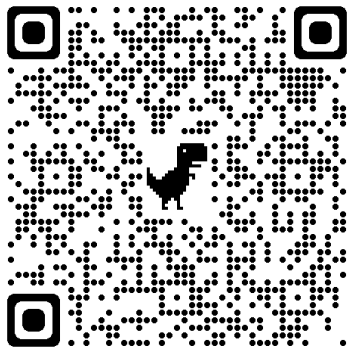
[실행결과]
KeyError: 'height'

```
if 'height' in my_dictionary[3]:
    data[5][1] = my_dictionary[3]['height']
else:
    print('Patient 3 does not exist height.')
if 'weight' in my_dictionary[3]:
    data[5][1] = my_dictionary[3]['weight']
else:
    print('Patient 3 does not exist weight.')
```

[실행결과]
Patient 3 does not exist height.
Patient 3 does not exist weight.

4. 조건문

◦ if 조건문 실습



```
data = [
    [1, 170, 100, 130, 120, 100, None],
    [2, 160, 80, 110, 100, 105, None],
    [3, 150, 40, 100, 70, 80, None]
]
diseasePatients = [2]

data[0][6] = 1 if data[0][0] in diseasePatients else 0
data[1][6] = 1 if data[1][0] in diseasePatients else 0
data[2][6] = 1 if data[2][0] in diseasePatients else 0
print(data)
```

[실행결과]
[[1, 170, 100, 130, 120, 100, 0], [2, 160, 80, 110, 100, 105, 1], [3, 150, 40, 100, 70, 80, 0]]

5. 반복문

◦ while 문

- 조건문이 참일 경우 수행

- 제어를 위한 별도의 변수가 필요함

```
result = 0
i = 0

while i <= 100:
    result += i
    i += 1
print(result)
```

[실행결과]
5050

```
result = (1 + 100)*50
print(result)
```

[실행결과]
5050

```
result = sum(range(101))
print(result)
```

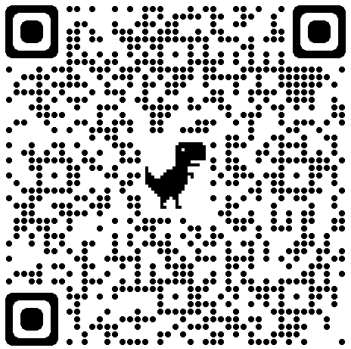
[실행결과]
5050

5. 반복문

◦ continue 문

- 인접한 반복문으로 건너뜀

```
data = [[1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 101], [1, 170, 100, 130, 120, 102], [1, 170, 100, 130, 120, 99], [1, 170, 100, 130, 120, 103], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 100], [2, 160, 80, 110, 100, 103], [2, 160, 80, 110,100, 100], [3, 150, 40, 100, 70, 80], [3, 150, 40, 100, 70, 85], [3, 150, 40, 100, 70, 82]]
i = 0
count = 0
while i < len(data):
    if data[i][0] != 1:
        i += 1
        continue
    count += 1
    i += 1
print(count)
```



[실행결과]
6

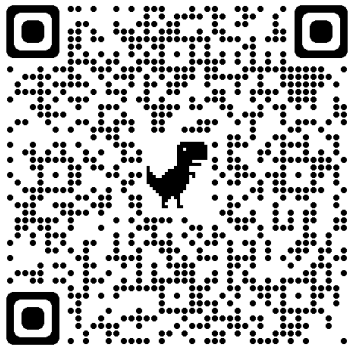
5. 반복문

◦ break 문

- 인접한 반복문을 중지시킴

- 조건문이 복잡하거나, 여러 조건 중 하나만 만족하면 탈출하도록 하기 위함

```
num = 7
i = 2
while True:
    if num % i == 0:
        break
    i += 1
if i == num :
    print(num,'는 소수입니다.')
```



[실행결과]
7 는 소수입니다.

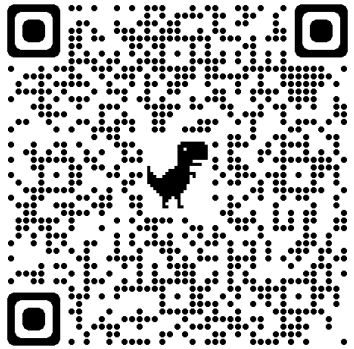
5. 반복문

◦ 반복문 내 else 문

- 반복문에서 break문을 통해 탈출했는지 검사

```
num = 7
i = 2
while i < num:
    if num % i == 0:
        break
    i += 1
else:
    print(num,'는 소수입니다.')
```

[실행결과]
7 는 소수입니다.



5. 반복문

◦ for 문

- 다른 프로그래밍 언어에서의 for 문과 상이함

- 반복할 변수 내에 있는 값들을 복사하여 반복함

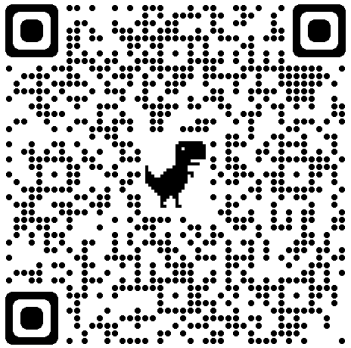
- range 함수를 활용하여 반복 횟수를 지정

5. 반복문

◦ for 문 실습

```
data = [
    [1, 170, 100, 130, 120, 100],
    [2, 160, 80, 110, 100, 105],
    [3, 150, 40, 100, 70, 80]
]

for p in data:
    print(p)
    p[0] = 0
print(data)
```



[실행결과]
[1, 170, 100, 130, 120, 100]
[2, 160, 80, 110, 100, 105]
[3, 150, 40, 100, 70, 80]
[[0, 170, 100, 130, 120, 100], [0, 160, 80, 110, 100, 105], [0, 150, 40, 100, 70, 80]]

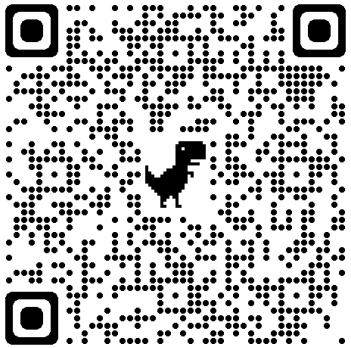
5. 반복문

◦ for 문 실습

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

for p in data:
    p = 0
print(data)
```

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



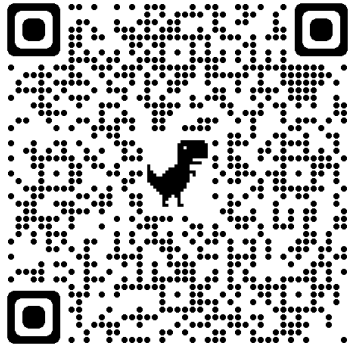
5. 반복문

◦ for 문 실습

```
data = [[1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 101], [1, 170, 100, 130, 120, 102], [1, 170, 100, 130, 120, 99], [1, 170, 100, 130, 120, 103], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 100], [2, 160, 80, 110, 100, 103], [2, 160, 80, 110,100, 100], [3, 150, 40, 100, 70, 80], [3, 150, 40, 100, 70, 85], [3, 150, 40, 100, 70, 82]]

list_each_patients = []

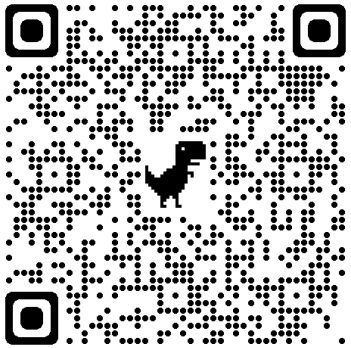
last = None
tmp = None
for p in data:
    if last == None:
        last = p[0]
        tmp = []
    if last == p[0]:
        tmp.append(p)
    else:
        list_each_patients.append(tmp)
        last = p[0]
        tmp = []
if len(tmp) > 1:
    list_each_patients.append(tmp)
```



5. 반복문

◦ for 문 실습

```
data = [[1, 170, 100, 130, 120, 100],
    [1, 170, 100, 130, 120, 103],
    [1, 170, 100, 130, 120, 105],
    [2, None, None, 110, 100, 105],
    [2, None, None, 110, 100, 102],
    [2, 160, 80, 110, 100, 103],
    [3, None, None, 100, 70, 80],
    [3, None, None, 100, 70, 85],
    [3, None, None, 100, 70, 90]]
print(data)
```

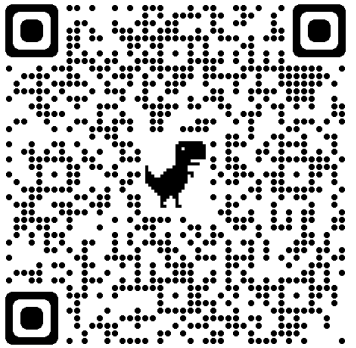


[실행결과]
[[1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 103], [1, 170, 100, 130, 120, 105], [2, None, None, 110, 100, 105], [2, None, None, 110, 100, 102], [2, 160, 80, 110, 100, 103], [3, None, None, 100, 70, 80], [3, None, None, 100, 70, 85], [3, None, None, 100, 70, 90]]

5. 반복문

- for 문 실습

```
my_dictionary = {}
for p in data:
    if p[0] not in my_dictionary:
        my_dictionary[p[0]] = {}
        if p[1] is not None:
            my_dictionary[p[0]]['height'] = p[1]
        if p[2] is not None :
            my_dictionary[p[0]]['weight'] = p[2]
    if 'height' not in  my_dictionary[p[0]] or my_dictionary[p[0]]['height'] is None:
        my_dictionary[p[0]]['height'] = p[1]
    if 'weight' not in  my_dictionary[p[0]] or my_dictionary[p[0]]['weight'] is None:
        my_dictionary[p[0]]['weight'] = p[2]
```



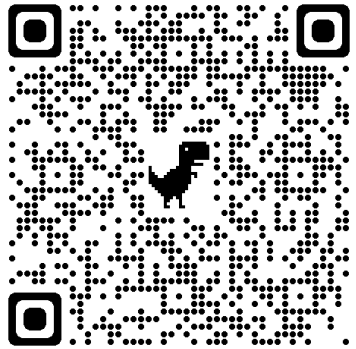
5. 반복문

◦ for 문 실습

```
print(my_dictionary)
```

[실행결과]
{1: {'height': 170, 'weight': 100}, 2: {'height': 160, 'weight': 80}, 3: {'height': None, 'weight': None}}

```
for p in data:
    if p[1] is None:
        if p[0] in my_dictionary and my_dictionary[p[0]]['height'] is not None:
            p[1] = my_dictionary[p[0]]['height']
    if p[2] is None:
        if p[0] in my_dictionary and my_dictionary[p[0]]['weight'] is not None:
            p[2] = my_dictionary[p[0]]['weight']
print(data)
```



[실행결과]
[[1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 103], [1, 170, 100, 130, 120, 105], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 102], [2, 160, 80, 110, 100, 103], [3, None, None, 100, 70, 80], [3, None, None, 100, 70, 85], [3, None, None, 100, 70, 90]]

6. 고급 문자열

◦ 문자열에서의 연산자

- + : 두 문자열을 연결함
- * : 문자열을 정해진 숫자만큼 반복
- <, <=, >, >= : 문자열 내 문자들을 알파벳 순서와 비교
- in : 문자열 전체가 문자열에 포함되어 있으면 참

6. 고급 문자열

◦ 인덱스

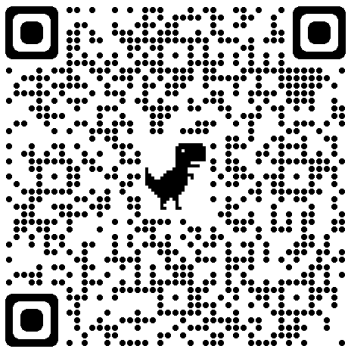
- 양의 인덱스와 음의 인덱스를 통해 접근가능

◦ 슬라이싱

- 문자열을 이용하여 부분 문자열을 만듬
- string[begin:end:step] : begin 생략 시 0, step 생략 시 1, begin부터 end 이전까지의 부분 문자열을 만듦

```
string = "Hello world"
print(string[-3:-1])
print(string[-1:-3:-1])
```

[실행결과]
rl
dl



6. 고급 문자열

- 대소문자 변환
  - lower : 모두 소문자로 변경된 문자열을 생성
  - upper : 모두 대문자로 변경된 문자열을 생성
- 문자열 검색
  - startswith : 접두사로 시작하면 참
  - endswith : 접미사로 문자열이 끝나면 참
  - find : 부분 문자열을 찾고 찾은 인덱스를 반환. 찾지 못한 경우 -1 반환
  - rfind : find와 유사하나 문자열 끝에서부터 찾음

```
from requests import get
pathName = get('http://172.28.0.2:9000/api/sessions').json()[0]['name']
pathName = pathName[:pathName.rfind('.')]
print(pathName)
```

6. 고급 문자열

◦ 문자열 분할

- split : 문자열을 주어진 구분자로 나누어 분할함

```
1 from requests import get
2 pathName = get('http://172.28.0.2:9000/api/sessions').json()[0]['name']
3 pathName = pathName[:pathName.rfind('.')]
4
5 split_string = pathName.split('_')
6
7 measureInterval = int(split_string[4])
8 unitsize = int(split_string[7])
9 lookback = int(split_string[2])
```

6. 고급 문자열

◦ 문자열의 주요 함수

- join : 리스트를 구분자로 결합
- len : 문자열의 길이를 반환
- reversed : 문자열의 리스트들을 알파벳의 역순으로 정렬
- sorted : 문자열의 리스트들을 알파벳 순으로 정렬
- replace : 기존 문자열을 다른 문자열로 대체함

6. 고급 문자열

◦ 문자열의 그 외 함수

- isalnum
- isalpha
- isdecimal
- isdigit
- isidentifier
- islower
- isprintable
- isspace
- istitle
- isupper
- title
- swapcase
- min
- max
- count
- index
- strip
- lstrip
- rstrip

6. 고급 문자열

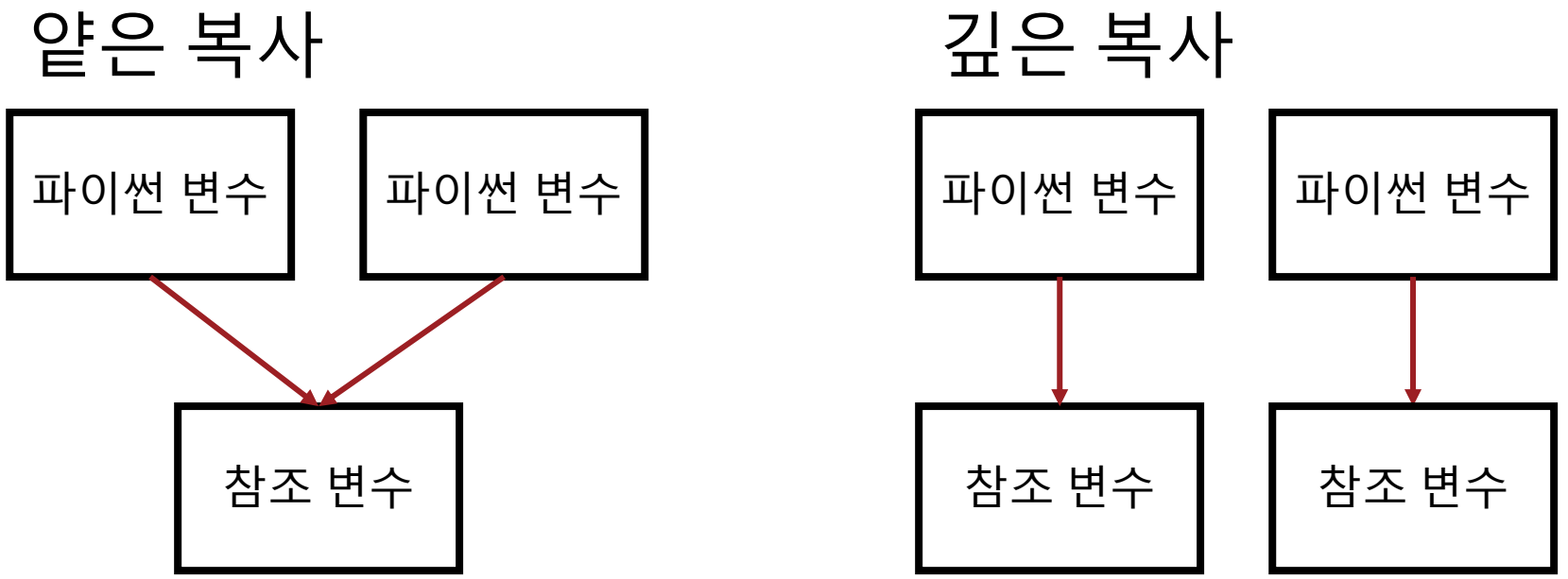
◦ 정규표현식

- 파이썬 스킬 업 기초를 넘어서, 파이썬을 파이썬답게 사용하자!, 브라이언 오버랜드, 존 베넷 저/조인석 역, 길벗
  - ➢ 6장, 7장

- re 패키지 검색

7. 고급 리스트

- 리스트 깊은 복사
  - 얕은 복사와 깊은 복사



7. 고급 리스트

◦ 리스트 깊은 복사

- 얕은 복사

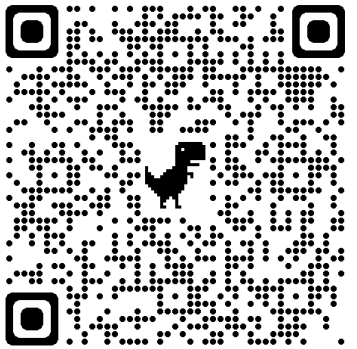
```
list_a = [1, 2, 3, 4, 5]
list_b = list_a
list_b[0] = 0
print(list_a)
```

[실행결과]
[0, 2, 3, 4, 5]

- 깊은 복사

```
import copy
list_b = copy.deepcopy(list_b)
list_b[0] = 3
print(list_a)
print(list_b)
```

[실행결과]
[0, 2, 3, 4, 5]
[3, 2, 3, 4, 5]



7. 고급 리스트

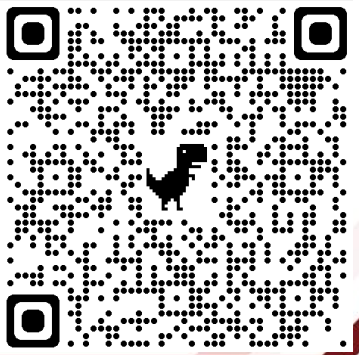
◦ 슬라이싱

- 문자열의 슬라이싱과 유사하게 부분 리스트를 생성함

- list[begin:end:step] : begin 생략 시 0, step 생략 시 1, begin부터 end 이전까지의 부분 리스트를 만듦

```
data = [[1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 101], [1, 170, 100, 130, 120, 102], [1, 170, 100, 130, 120, 99], [1, 170, 100, 130, 120, 103], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 100], [2, 160, 80, 110, 100, 103], [2, 160, 80, 110,100, 100], [3, 150, 40, 100, 70, 80], [3, 150, 40, 100, 70, 85], [3, 150, 40, 100, 70, 82]]

list_each_patients = []
list_each_patients.append(data[0:6])
list_each_patients.append(data[6:11])
list_each_patients.append(data[11:14])
```



7. 고급 리스트

◦ 슬라이싱

```
print(list_each_patients[0])
```

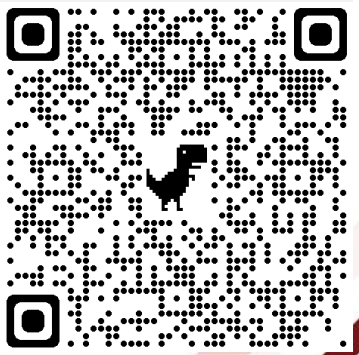
[실행결과]
[[1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 100], [1, 170, 100, 130, 120, 101], [1, 170, 100, 130, 120, 102], [1, 170, 100, 130, 120, 99], [1, 170, 100, 130, 120, 103]]

```
print(list_each_patients[1])
```

[실행결과]
[[2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 105], [2, 160, 80, 110, 100, 100], [2, 160, 80, 110, 100, 103], [2, 160, 80, 110, 100, 100]]

```
print(list_each_patients[2])
```

[실행결과]
[[3, 150, 40, 100, 70, 80], [3, 150, 40, 100, 70, 85], [3, 150, 40, 100, 70, 82]]



7. 고급 리스트

◦ 리스트의 주요 함수

- append : 리스트에 항목 추가
- len : 리스트의 길이를 반환
- min : 리스트 내 최소 값 반환
- max : 리스트 내 최대 값 반환
- reversed : 리스트를 내림차순으로 정렬함
- sorted : 리스트를 오름차순으로 정렬함

7. 고급 리스트

◦ 리스트의 그 외 함수

- sum
- clear
- extend
- insert
- remove
- count
- index
- pop
- sort
- reverse