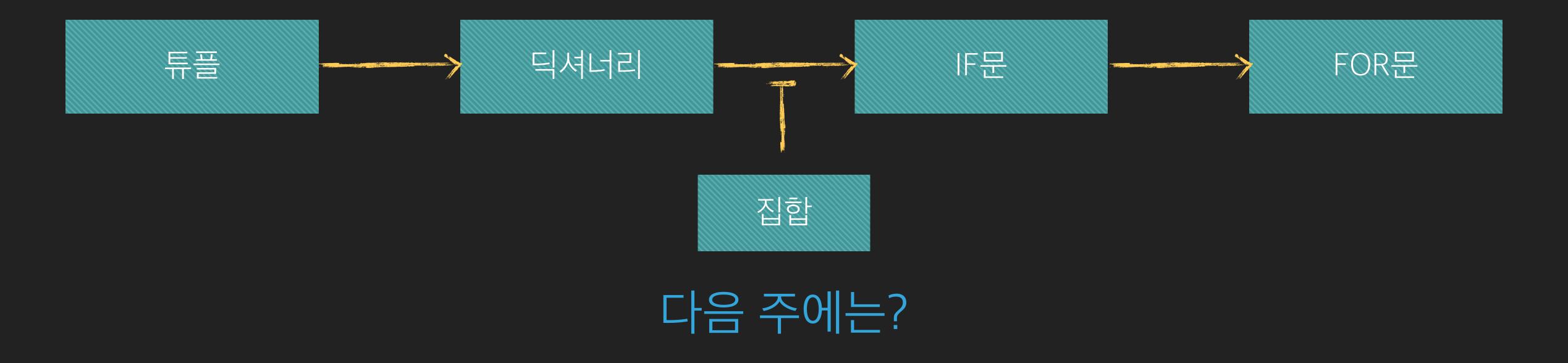
국민대학교 인공지능 연구실

파이썬기초2주차

2주차 목차



WHILE문 ---> 함수 ---> 모듈 ---> 객체

INTRO

- ▶ 1주차 colab 실습 링크
 - https://colab.research.google.com/drive/1wgHQKrv2Q6bAjhfKZFItB785nGuEVw-Q
- ▶ 2주차 colab 실습 링크
 - https://colab.research.google.com/drive/1dQ9VYzAuNrHVtF2XOshEztpToIZMF0hz

튜플(TUPLE)

- ▶ 변경이 불가능한 리스트
 - ▶ 추가, 수정, 삭제가 불가능함
- ▶ 리스트는 []으로 둘러싸지만 튜플은 ()로 둘러쌈
- ▶ 값을 변화시킬 수 없다는 점만 제외하면 리스트와 완전히 동일함
 - ▶ 리스트와 동일하게 인덱싱, 슬라이싱, 더하기, 곱하기, 길이구하기 모두 가능함
- ▶ 스스로 정의할 때에는 수정 가능한 list형을 사용하면 간편하지만, '함수'의 출력이 여러 개인 경우 자동으로 tuple로 정리됨
- ▶ 또한 기존 함수의 입력이 tuple형으로 지정된 경우도 있음 ─> 자료형을 tuple로 바꿔준 후 입력으로 사용

- ▶ 사전에 'people': '사람들', 'baseball': '야구' 처럼 한 쌍을 이루는 자료형
 - ▶ 딕셔너리는 key와 value를 한 쌍으로 가짐
 - ▶ 리스트나 튜플처럼 순차적으로 해당 요솟값을 구하지 않고 key를 통해 value를 얻음
 - ▶ 때문에 데이터의 위치가 아닌 key가 중요함
 - ▶ 즉, people의 뜯을 찾기 위해서 사전의 내용을 순차적으로 보는 것이 아니라 people이라는 단어 가 있는 곳만 펼쳐보는 것!
- ▶ 딕셔너리의 모습
 - Key1: Value1, Key2: Value2, Key3: Value3}

- ▶ 딕셔너리에는 value값으로 리스트를 넣을 수 있음
 - \rightarrow Ex) a = {'a' : [1, 2, 3]}
- ▶ 딕셔너리에서 쌍을 추가하고 삭제하는 법은 다음과 같다
 - ▶ a라는 딕셔너리가 있을 때, 추가하고 싶다면 a[3] = 5 라고 입력한다면 key와 value가 각각 2와 5인 쌍이 추가됨
 - ▶ 삭제하고 싶다면 1) del a[3] 이라고 하면 key값이 3인 {key:value} 쌍이 제거됨
 - ▶ 딕셔너리에서는 []안에 인덱스가 들어가는 것이 아니기 때문에 없는 key값을 넣게 되면 에러 발생!
 - ▶ 2) a.pop(3)이라고 한다면 key값으로 딕셔너리에서 찾은 값을 반환한 다음 삭제함

- dic1 = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
- ▶ key값을 이용해서 value값 얻기
 - ▶ nameOfDict[Key]를 통해서 value값을 얻을 수 있음 ─> 없으면 에러 발생
 - Ex) dic1['name'] —> pey
 - ▶ nameOfDict.get(key)를 통해서 value값을 얻을 수 있음 ─> 없으면 None 리턴
 - Ex) dic1.get('name')
- ▶ Key 리스트 만들기
 - ▶ nameOfDict.key()는딕셔너리의 key값만 모은 dict_keys객체를 리턴
 - Ex) dic1.keys() —> dict_keys['name', 'phone', 'birth'])

- dic1 = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
- ▶ Value 리스트 만들기
 - ▶ nameOfDict.values()는딕셔너리의 value값만 모은 dict_values객체를 리턴
 - Ex) dic1.values() —> dict_values(['pey', '0119993323', '1118'])
- key와 value 쌍 얻기
 - ▶ nameOfDict.items()는 key와 value의 쌍을 튜플로 묶은 값을 dict_items로 리턴
 - Ex) dic1.items = dict_items([('name', 'pey'), ('phone', '0119993323'), ('birth', '1118')])

- dic1 = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
- ▶ 특정 key가 딕셔너리 안에 있는지 조사하기
 - key in nameOfDict 를 통해서 해당 key가 딕셔너리 안에 있는지 조사할 수 있음
 - ▶ 만약 있다면 True, 없다면 False 리턴
 - Ex) 'name' in dic1 —> True
- ▶ key와 value값 모두 지우기
 - nameOfDict.clear()를 통해서 딕셔너리 안의 모든 요소를 삭제할 수 있음
 - Ex) dic1.clear()

집합(SET)

- > 중복을 허용하지 않으며 순서가 없음
 - ▶ 리스트나 튜플은 순서가 있기 때문에 인덱싱을 통해 자료형의 값을 얻을 수 있지만 집합은 순서가 없기 때문에 인덱싱으로 값을 얻을 수 없음
 - ▶ 만약 집합에 저장된 값을 인덱싱으로 접근하려면 리스트나 튜플로 변환한 후 해야 함
- ▶ 집합을 사용해서 교집합, 합집합, 차집합을 구할 수 있음
 - ▶ s1=set([1, 2, 3, 4, 5, 6]). s2=set([4, 5, 6, 7, 8, 9])
 - ▶ 교집합 (s1 & s2) —> {4, 5, 6} | s1.intersection(s2) | s2.intersection(s1)
 - ▶ 합집합 (s1 | s2) —> {1, 2, 3, 4, 5, 6, 7, 8, 9} | s1.union(s2) | s2.union(s1)
 - ▶ 차집합 (s1 s2) -> {1, 2, 3} | s1.difference(s2) | s2.difference(s1)

집합(SET)

- ▶ 원소 추가
 - setName.add(원소) —> 원소 한개를 추가 가능
 - > setName.update([원소1, 원소2, 원소3]) -> 집합에 리스트에 있는 원소들을 추가 가능
- ▶ 원소 제거
 - > setName.remove(원소) -> 원소가 집합에 있으면 삭제하고 없으면 에러를 발생
 - > setName.discard(원소) -> 원소가 집합에 있으면 삭제

- bool 자료형
 - ▶ 참(True) 혹은 거짓(False)을 나타내는 자료형
 - ▶ 조건에 대해서 참인 경우에는 'True' 리턴, 거짓일 경우에는 'False' 리턴
- ▶ 주어진 조건을 판단한 후 상황에 맞게 처리해야할 경우에 사용함
 - ▶ 만약 · · · 하다면(if)
 - ▶ 조건문의 참(True)와 거짓(False)를 판별한 후 참(True)인 경우에만 실행문을 실행함
 - ▶ 조건문을 만들 때는 if 조건문: 바로 아래 문장부터 if문에 속하는 모든 문장에 들여쓰기를 해주어야 함

수행할 문장1 수행할 문장2 if 조건문:

if 조건문:

수행할 문장3

실행문

elif 조건문:

실행문

else:

실행문

- ▶ else문
 - ▶ 위의 조건이 모두 거짓인 경우에 수행
 - ▶ 들여쓰기로 구분
- ▶ 들여쓰기
 - ▶ 코드블록은 들여쓰기로 결정됨
 - ▶ 동일한 블록은 같은 거리만큼 들여쓰기
 - ▶ 4칸씩 들여쓰기를 권장함
- ▶ 비교연산자
 - ▶ 비교연산자(<, >, ==, !=, <=, >=)를 사용해서 특정 조건이 참인지 거짓인지를 판단할 수 있음
 - ▶ 조건이 참일 경우에만 조건문 내부의 실행문을 실행함

if <조건1>:
 <조건1이 참일 때 실행하는 문장>
else:
 <조건1이 거짓일 때 실행하는 문장>

- ▶ 다중 조건 테스트
 - and: 두 조건이 동시에 만족되어야 True
 - or: 두 조건 중 하나만 만족되면 True
 - not: 조건의 참/ 거짓 값의 반대
- 'in' and 'not in'
 - ▶ 리스트, 튜플, 문자열 안에 특정 값이 있는지 확인할 수 있음
 - ▶ 사용법: x in s | x not in s 의 형태로 활용할 수 있음
 - Ex) 1 in [1, 2, 3] | 'a' in ('a', 'b', 'c') | 'h' in 'hello'

- ▶ 다중 조건 판단 elif
 - ▶ if와 else만으로는 다양한 조건을 판단하기가 어렵기 때문에 다중 조건 판단을 사용함
- ▶ 삼항 연산자(Ternary operator)
 - ▶ 형식: <조건문이 참인 경우> if <조건문> else <조건문이 거짓인 경우>
 - ▶ if문을 대신하여 위와 같은 형식으로도 작성할 수 있지만 가독성이 너무 떨어지면 사용하지 않는 편이 좋음

if 조건문:

실행문

elif 조건문:

실행문

elif 조건문:

실행문

else:

실행문

FOR문

- ▶ 컴퓨터가 잘하는 작업 ─> 반복하는 것!
- ▶ Loop : 반복 작업(계산)
 - ▶ Counting loop = 특정 횟수만큼 반복하는 것
 - ▶ Conditional loop = 특정한 조건이 만족될 때 까지 반복
- for loops
 - ▶ 시퀀스 데이터로는 리스트, 튜플, range를 사용할 수 있음
- range
 - ▶ range(num1, num2, num3) —> 'num1부터 num2-1까지의 숫자까지 num3간격으로'
 - range(1, 10, 2) —> 1, 3, 5, 7, 9

for 타깃 in 시퀀스:

실행문

FOR문

- continue사용법
 - ▶ 반복문을 수행하는 과정에서 맨 처음으로 다시 돌아가게 만들고 싶을 때 사용
 - > 즉, 특정 조건에서 아무것도 일어나지 않게 하고 싶을 때 사용함
- break사용법
 - ▶ 반복문을 수행하는 과정에서 강제로 반복문의 수행을 멈추고 싶을 때 사용
 - ▶ break을 사용하면 반복을 더 이상 수행하지 않고 반복문을 빠져나가게 됨