|  |  |
| --- | --- |
| 교육 제목 | **재귀함수, 튜플 람다 with 구문, 파일처리, 제너레이터 이터레이터, 예외처리** |
| 교육 일시 | 2021년 11월 02일 |
| 교육 장소 | 영우글로벌러닝 |
| **교육 내용** | |
| 내용 | 2021.11.02 7일차     1. 재귀함수    1. 재귀함수 / 메모화 / 조기리턴    2. 반복문으로 팩토리얼 구하기    3. for i in range(1,n+1)   output \*= i   1. 재귀함수로 팩토리얼 구하기 3. 메모화**(딕셔너리 활용)**    1. 재귀함수의 값을 딕셔너리에 저장하고, 다시 연산할 필요 없이 딕셔너리에서 값을 가져오는 방식. 연산 횟수가 크게 줄어든다.       1. global 변수 : 변수 앞에 global 을 붙이면 함수 밖에서도 변수를 호출할 수 있음.          1. 각 함수를 벗어나서도 같은 변수를 사용할 수 있음. 4. 조기리턴    1. : 조기에 리턴을 하여 이후 처리문을 실행하지 않는 기술.   들여쓰기를 줄일 수 있고, 들여쓰기가 적을 수록 좋은 코드로 본다.    2. 튜플 / 람다 / with 구문    1. 튜플 : 함수와 함께 많이 사용되는 리스트와 비슷한 자료형으로, 한번 결정된 요소를 바꿀 수 없다는 점이 리스트와 다름. 튜플 자체를 바꾸는 것은 가능하나, 각각의 요소를 변경할 수없음.       1. **한번 결정된 요소는 바꿀 수 없음.**       2. (데이터, 데이터, 데이터, …)       3. 괄호 없는 튜플 => tuple = 10, 20, 30, 40   (괄호가 있으나 없으나 같다.)          1. a, b, c = 10, 20, 30    -> 튜플은 값 변경 불가.       4. 튜플의 활용          1. 요소의 값을 교환하기             1. a, b = 10 ,20 (이렇게 지정 가능)             2. a,b = b, a    -> 값이 서로 바뀐다. (이렇게도 응용 가능)      1. 람다(lambda) : 매개변수로 함수를 전달하기 위해 함수 구문을 작성하는 것이 번거롭기때문에 사용함    1. 기능을 매개변수로 전달하는 코드를 더 효율적으로 작성    2. map(함수, 리스트)       1. 리스트에 있는 요소 하나하나에 대해서 함수에 매개변수로 넣어 연산한다.       2. list = [1, 2 ,3]       3. map(power,list) = [1, 4, 9]    3. filter(함수, 리스트)       1. 리스트에 있는 요소 하나하나에 대해서 True인 요소를 return한다.       2. filter(under\_3,list) = [1, 2]    4. lambda 매개변수 : 리턴값  (람다를 활용한 함수의 정의       1. power = lambda x : x \* x       2. under\_3 = lambda x : x < 3 2. with 구문 3. 파일 처리    1. 파일 열기(open) - 파일읽기(read) - 파일쓰기(write)       1. # file open, write, close       2. # file = open('경로명/파일명',mode: 'w, r, a')       3. # 현재 디렉토리  **= ./**       4. # 상위 디렉토리 **= ../**       5. # 현재경로에서 하위 디텍토리 **= ./디렉토리명**    2. open() 함수       1. 파일 객체 = open(문자열 : 파일경로, 문자열 : 읽기모드)       2. 모드 w / a / r    3. close() 함수       1. 파일 객체.close()    4. with : 문장종료하면 file.close() 자동실행 4. 제너레이터 -> yield / next()    1. 이터레이터를 직접 만들 때 사용하는 코드    2. 함수내부에 yield 키워드 사용하면 해당 함수는 제너레이터 함수가 됨.    3. 일반 함수와 달리 호출해도 함수내부 코드가 실행되지않음.    4. for 반복문은 next()함수로 실행된다고 보면 됨.     7. 이터레이터 / 제너레이터 함수를 사용하게되는 이유 5. 예외 처리    1. 오류와 예외       1. 구문오류 : 프로그램이 실행되기전에 발생하는 오류       2. 런타임 오류(예외) : 프로그램이 실행되면서 발생하는 오류    2. 예외를 처리하는 방법       1. 조건문을 사용하는 방법       2. try 구문을 사용하는 방법          1. try:    -> 예외가 발생할 가능성이 있는 코드          2. except : -> 예외가 발생할 경우 실행할 코드          3. else: -> 예외가 발생하지않을 경우 실행할 코드          4. finally: -> 무조건 실행하는 코드          5. except / pass 키워드 조합 : 예외가 생길경우 무시    3. Exception : 모든 예외의 어머니       1. except Exception as 변수 : 모든 예외의 경우 실행       2. except ValueError : 정수로 변환될 수없는 값 입력시 실행       3. except IndexError : 범위를 벗어난 경우에 실행    4. 강제로 에러 발생시키기       1. raise NotImplementedError -> 에러 발생    5. 재귀함수 메모화 새롭게 짜본 코드 |