|  |  |
| --- | --- |
| 교육 제목 | Python |
| 교육 일시 | 21.09.24 |
| 교육 장소 | 영우글로벌러닝 |
| **교육 내용** | |
| 오전 | 팩토리얼(factorial)  n! = n \* ( n - 1 ) \* ( n - 2 ) \* . . . \* 1  3! = 3 \* 2 \* 1    반복문으로 팩토리얼 구하기  재귀 함수로 팩토리얼 구하기  10!  == 10 \* 9 \* . . . \* 2 \* 1  == 10 \* 9!  == 10 \* 9 \* 8!    # 함수를 선언합니다.  def factorial(n) :  # 변수를 선언합니다.  output = 1  # 반복문을 돌려 숫자를 더합니다.  for i in range(1, n + 1):  output \*= i  # 리턴합니다.  return output    # 함수를 호출합니다.  print("1!:", factorial(1))  print("2!:", factorial(2))  print("3!:", factorial(3))  print("4!:", factorial(4))  print("5!:", factorial(5))    -------------------------    피보나치 수열  f(4) <- f(3) + f(2)  f(3) <- f(2) + f(1)    # 변수를 선언합니다.  counter = 0    # 함수를 선언합니다.  def fibonacci(n):  # 어떤 피보나치 수를 구하는지 출력합니다.  print("fibonacci({})를 구합니다.".format(n))  global counter  counter += 1  # 피보나치 수를 구합니다  if n == 1:  return 1  if n == 2:  return 2  else:  return fibonacci(n-1) + fibonacci(n-2)    # 함수를 호출합니다.  fibonacci(10)  print("---")  print("fibonacci(10) 계산에 활용된 덧셈 횟수는 {}번 입니다.".format(counter))    ---------------------------    메모 변수 함수를 사용  # 메모 변수를 만듭니다.  dictionary = {  1: 1,  2: 1  }    # 함수를 선언합니다.  def fibonacci(n):  if n in dictionary:  # 메모가 되어 있으면 메모된 값을 리턴  return dictionary[n]  else:  # 메모가 되어 있지 않으면 값을 구함  output = fibonacci(n-1)+fibonacci(n-2)  dictionary[n] = output  return output    # 함수를 호출합니다  print("fibonacci(10):", fibonacci(10))  print("fibonacci(20):", fibonacci(20))  print("fibonacci(30):", fibonacci(30))  print("fibonacci(40):", fibonacci(40))  print("fibonacci(50):", fibonacci(50))    --------------------------------------    조기 리턴  # 힘수를 선언합니다.  def fibonacci(n):  if n in dictionary:  # 메모되어 있으면 메모된 값을 리턴  return dictionary[n]  else:  # 메모되어 있지 않으면 값을 구함  output + fibonacci(n-1) + fibonacci(n-2)  dictionary[n] = output  return output    ------------------------    키보드에서 숫자를 입력받아 계산 하는 프로그램 작성    # 키보드에서 두개의 숫자 입력  # 두개의 숫자를 리스트로  input\_number = input().split  # 문자를 숫자로 변환  for( i , value in enumerate(input\_number):  input\_number[i] = int(value)  # 부호 입력  input\_char = input()  # 부호에 따른 계산식 구현  if input\_char == '+':  result = plus\_func(input\_number)    --------------------    # 평탄화 함수 작성  def faltten(data):  # 최종 결과를 저장할 리스트 작성  output = []  # 리스트 각각의 item을 가져와서 작업  for item in data:  # 데이터 타입이 item 이면 flatten함수 호출 아니면 output에 리스트 추가  if type(item) == list:  output += flatten(item)  else:  output.append(item)  return output    # 리스트가 포함된 리스트 작성  example = [[1,2,3], [4,[5,6]],7,[8,9]]  print(" example original : ", example)  print(" example flatten : ", flatten(example)) # 평탄화 함수 실행    -----------------    고급 함수  튜플 / 람다 / 파일 처리 <- 파이썬만의 특별 문법  제너레이터  튜플 : 함수와 함께 많이 사용되는 리스트와 비슷한 자료형으로, 한번 결정된 요소를 바꿀 수 없다는 점이 리스트와 다름  람다 : 매개변수로 함수를 전달하기 위해 함수 구문을 작성하는 것이 번거롭고 코드 낭비라 생각 될 때 함수를 간단하고 쉽게 선언하는 방법  tuple 과 lambda  튜플과 리스트는 ()와 [ ]로 구분  튜플은 () 리스트는 [ ]    #튜플( )  [a,b] = [10,20]  (c,d) = (30,40)  print(a,b,c,d)  a = 40  a  c = (1,2,3) # 값 변경 불가능  # c[0]=4  a,b,c = 10,20,30  print(a,b,c)    # 값을 교환  a,b = 10,20  print("before : ", a,b)  a,b = b,a  print("after : ", a,b)  ----------------    # 문제) 두 수를 입력받아 큰 수에서 작은 수(num1 - num2)의 형식으로 작성하세요.  # 함수 정의  def minus\_func(num1 , num2):  return (num1 - num2)    def trans\_func(num1, num2):  if num1 < num2:  return (num2, num1)  else:  return (num1, num2)    input\_numbers = input(" 두 수 입력 : ").split()  (num1, num2) = trans\_func(int(input\_numbers[0]),int(input\_numbers[1]))  print(minus\_func(num1,num2))    ----------------    람다 : 기능을 매개변수로 전달하는 코드를 더 효율적으로 작성  함수의 매개변수로 함수 전달  map(함수, 리스트) \*파이썬에서 중요한 함수  filter(함수, 리스트)    # 함수를 선언합니다.  def power(item):  return item \* item  def under\_3(item):  return item < 3    # 변수를 선언합니다.  list\_input\_a = [1,2,3,4,5]    # map() 함수를 사용합니다.  output\_a = map(power, list\_input\_a)  print("# map() 함수의 실행결과")  print("map(power, list\_input\_a):", output\_a)  print("map(power, list\_input\_a):", list(output\_a))  print()    # filter() 함수를 사용합니다.  output\_b = filter(under\_3, list\_input\_a)  print("#filter() 함수의 실행결과")  print("filter(under\_3, list\_input\_a):", output\_b)  print("filter(under\_3, list\_input\_a):", list(output\_b))    ----------------------    리스트 [ 1,2,3,4,5 ]  for item in 리스트:  함수명(item)  => map(함수명,리스트)  => fliter(함수명,리스트) : 함수에 조건식    ---------------------    튜플 -> 람다    # 함수를 선언합니다.  def power(item):  return item \* item  power = lambda x: x\*x # lambda 매개변수 : 리턴 값  # def under\_3(item):  # return item < 3  under = lambda x: x < 3    # 변수를 선언합니다.  list\_input\_a = [1,2,3,4,5]    # map() 함수를 사용합니다.  output\_a = map(power, list\_input\_a)  print("# map() 함수의 실행결과")  print("map(power, list\_input\_a):", output\_a)  print("map(power, list\_input\_a):", list(output\_a))  print()    # filter() 함수를 사용합니다.  output\_b = filter(under\_3, list\_input\_a)  print("#filter() 함수의 실행결과")  print("filter(under\_3, list\_input\_a):", output\_b)  print("filter(under\_3, list\_input\_a):", list(output\_b))    ---------------------    lambda 매개변수: 리턴 값  map(lambda x: x\*x, 리스트)    ------------------    파일처리  파일 열기(open) -> 파일 읽기(read) -> 파일 쓰기 (wirte)    # file을 이용한 데이터 저장  # open, read or write, close  # open(파일명, 모드)  # w-> 기존의 파일이 존재하면 기존 데이터 삭제 후 새로 생성, 존재하지 않으면 새로 생성  # a-> 마지막에 추가  # r-> 읽기만 가능, 존재하지 않으면 Error  #file = open("phython.txt","w") # file open  file = open("phython.txt","a") # file open  file.write("Hello python programming!!!!\n") # file write  file.close() # file close    # file.write("test") # Error 발생  file = open("phython.txt","r")  contents = file.read()  print("file read data : ", contents)    -------------------------    open( )함수를 사용했으면 반드시 close( ) 함수를 사용해야한다  단, with 함수를 사용하면 close를 사용하지 않아도 된다  with open(문자열: 파일 경로, 문자열: 모드) as 파일 객체 : 문장    # 100명의 키와 몸무게를 이름과 같이 파일에 저장하기  import random  # 한글 리스트 생성  hanguls = list("가나다라마바사아자차카타파하")  # 파일을 쓰기 모드로 엽니다.  with open("info.txt", "w") as file:  for i in range(100):  # 랜덤한 값으로 변수를 생성합니다.  name = random.choice(hanguls) + random.choice(hanguls)  weight = random.randrange(40, 100)  height = random.randrange(140, 200)  #텍스트를 씁니다.  file.write("{}, {}, {}\n".format(name, weight, height))    with open("info.txt","r") as file:  for line in file:  (name, height, weight) = line.split(',')  print("{}, {}, {}".format(name, height, weight))    -----------------------    제너레이터(generator)  이터레이터를 직접 만들 때 사용하는 코드  함수 내부에 yield 키워드 사용하면 해당 함수는 제너레이터 함수 됨  일반 함수와 달리 호출해도 함수 내부 코드가 실행되지 않음    # 제너레이터 함수 만들기 : yield 키워드를 넣으면 일반 함수가 제너레이터 함수가 됨  def test():  print("함수 호출 111")  yield 1  print("함수 호출 222")  yield 2  print("함수 호출 333")  yield "abc"    output = test()  print("시작 00 : ", next(output())  print("시작 11 : ", next(output())  print("시작 22 : ", next(output())  next(output()) # next() 함수를 사용해야 제너레이터 함수 호출    --------------------    p269 문제 풀이  numbers = list(range(1, 10 + 1)) # 1부터 10까지의 수를 리스트로    print("# 홀수만 추출하기")  print(list(filter(lambda x: x%2,numbers)))  print()    print("# 3 이상, 7 미만 추출하기")  print(list(filter(lambda x: 3 <= x < 7,numbers)))  print()    print("# 제곱해서 50 미만 추출하기")  print(list(filter(lambda x: x\*x < 50,numbers)))  print()    ------------------    예외 처리    구문 오류, 예외(런타임 에러), 기본 예외 처리, by except 구문  오류(Error)  구문 오류(syntax error)  런타임 오류(runtime error)    # 숫자를 입력 받습니다.  user\_input\_a = input("정수 입력> ")    # 사용자 입력이 숫자로만 구성되어 있을 때  if user\_input\_a.isdigit():  # 숫자로 변환합니다.  number\_input\_a = int(user\_input\_a)  # 출력합니다.  print("원의 반지름:", number\_input\_a)  print("원의 둘레:", 2\* 3.14 \* number\_input\_a)  print("원의 넓이:", 3.14 \* number\_input\_a \* number\_input\_a)  else:  print("정수를 입력하지 않았습니다.")        # 조건문으로 예외 처리하기  PI=3.14  # 예외처리 try : 실행할 구문 입력 except : 예외 처리 구문 입력  try :  input\_number = int(input(" 정수 입력> ")) # 예외가 발생 가능 코드    print("원의 반지름 : ", input\_number)  print("원의 둘레: ", input\_number\*2\*PI)  print("원의 넓이 : ", input\_number\*input\_number\*PI)  except:  print("정수를 입력하지 않음") |
| 오후 | try except  try : 예외가 발생할 가능성이 있는 코드  except : 예외가 발생했을 때 실행할 코드  else : 예외가 발생하지 않았을 때 실행할 코드    # try  PI = 3.14  try :  input\_number = int(input(" 정수 입력> ")) # 예외가 발생 가능 코드    except:  print("정수를 입력하지 않음")  else:  print("원의 반지름 : ", input\_number)  print("원의 둘레: ", input\_number\*2\*PI)  print("원의 넓이 : ", input\_number\*input\_number\*PI)  finally:  print:("무조건 실행")    try + except  try + except + else  try + except + finally  try + except + else + finally  try + finally    ---------------    def write\_func(filename, test):  try:  file = open(filename, "w")  return # rereturn이 중간에 존재  file.write(text) # 실행 안됨  except:  print("file error")  finally: # 중간에 return이 되어도 무조건 실행  print("try and . . .")  file.close()    write\_func("file\_close.txt","Hello file")    ---------------    # 키보드에서 파일명을 입력받아 "r"모드로 열어서 파일의 내용을 화면에 출력  # 없으면 "w"모드로 open후 "new file"을 파일에 저장 후 파일 close  # try ~ except ~ finally 활용  # 파일명 입력    filename = input("파일명 입력 : ")  try:  file = open(filename, "r")  print(file.read())  except Exception as err:  print("file open err : ",err)  file = open(filename,"w")  file.write("New file")  finally:  file.close()    print("file.closed : ",file.closed)    -------------    구문 오류 : 프로그램의 문법적 오류로 프로그램이 실행조차 되지 않게 만드는 오류  예외(런타임 에러) : 프로그램 실행 중에 발생하는 오류  기본 예외 처리 : 조건문 등을 사용해 예뢰를 처리하는 기본적인 방법  try except 구문 : 예외 처리에 특화된 구문    ------------    # 변수를 선언합니다.  list\_number = [52, 273, 32, 72, 100]    # try except 구문으로 예외를 처리합니다.  try:  # 숫자를 입력받습니다.  number\_input = int(input("정수 입력> "))  # 리스트의 요소를 출력합니다.  print("{}번째 요소: {}".format(number\_input, list\_number[number\_input]))  예외.발생해주세요()  except ValueError as exception:  # ValueError가 발생하는 경우  print("정수를 입력해 주세요!")  print(type(exception), exception)  except IndexError as exception:  # IndexError가 발생하는 경우  print("리스트의 인텍스를 벗어났어요!")  print(type(exception), exception)    ---------------    raise구문    number = 10    if number > 0:  pass  else:  raise NotImplementedError    ------------------    '''파일명을 입력받아 "w" 모드로 open한 후  이름과 성적을 입력받아 파일에 저장  이름에 "end"가 입력되면 file을 close한 후  "r"모드로 파일을 다시 open  파일에서 자료를 읽어 list에 저장한 후  키보드로부터 검색할 이름을 검색 한 후 있으면 이름과 성적 출력  없으면 " Not found " 출력  파일을 open하는 함수 file\_open()작성'''    # 파일명 입력  # 파일 open 함수 호출  # 자료 입력 ( 이름, 성적), 이름에 "end"입력되면 입력 종료  # 파일 close  # 파일 open 함수 호출  # 자료를 가져와서 변수에 저장  # 검색 할 이름 입력  # 자료 검색 : 존재하면 출력, 없으면 " Not found" 출력    # 파일명 입력  file\_name = input("파일명 입력 > ")  file = file\_open(file\_name,"a") # 파일 오픈 함수 호출  file\_write(file) # 파일에 저장하는 함수 호출  file.close() # 파일 Close    # 파일에서 자료를 가져와서 변수에 저장  names, scores = [], []  with file\_opne(file\_name,"r") as file:  for item in file:  values = item.split(',')  names.append(values[0])  scores.append(int(values[1]))    # 검색할 이름 입력{  name = input("find name > ")  # 자료 검색 : 존재하면 출력, 없으면 " Not found " 출력  if name in names:  print("{} : {}".format(name,scores[names.index(name)]))  else:  print("Not found")    # 파일open  def file\_open(file\_name, file\_mode):  file = open(file\_name,file\_mode)  return file    def file\_write(file): # 자료 입력 ( 이름, 성적), 이름에 "end"입력되면 입력 종료  while True:  name = input(" 이름 입력 : 'end' 입력시 종료 > ")  if name == 'end':  break    while True:  try:  score = int(input("성적 입력 > "))  break  except:  print("숫자 입력")  context = name + ',' + str(score) + '\n'  file.write(context) # file에 자료 저장    -------------    함수의 활용 : 제귀함수 - 메모리  제너레이터  map(함수명, 리스트)  filter(함수명, 리스트)  lambda 매개변수: 리턴값  file 처리하는 함수 open, close, read, write  try: ~ except: ~ else: ~ finally:  phython : 자료의 형태 - 숫자,문자,불리언  list[ ], dictionary{ }  튜플( )  list[ 반복문 ] : 리스트 내포  조건식 if ~ elif ~ else  반복문 for(변수 in 반복자료):  while  while 빠져나가기 -> break, countinue    fun\_a(10,20) -> def func(a,b) : a,b가 매개변수 |