목차

[**#lay the goround python\_1** 2](#_Toc37333877)

[**#lay the goround python\_2** 7](#_Toc37333878)

[**#lay the goround python\_3** 9](#_Toc37333879)

[**#lay the goround python\_4** 13](#_Toc37333880)

[**#lay the goround python\_5** 19](#_Toc37333881)

# **#lay the goround python\_1**

1. **숫자형**
   1. **integer : 정수를 뜻하는 자료형**  
      a = 10
   2. **float : 실수형의 소수점 표현 방식**a = 3.14e2 #컴퓨터식 지수 표현 방식
   3. **8진수(Octal)** 0o 또는 0O(숫자 0 + 알파벳 소문자 o 또는 대문자 O)로 시작  
      a = 0o177
   4. **16진수(Hexdecimal)** 0x로 시작  
      a = 0x8ff
   5. **숫자 연산자**사칙연산 : + - \* /  
      x의y 제곱 : x \*\* y  
      나눗셈 후 나머지를 반환 : %  
      나숫셈 후 몫을 반환 : //
2. **문자열 자료형**
   1. **string : 문자열**print('"문자열 안에 "큰따옴표" 가 포함 됨')  
      print("문자열 안에 따옴표's가 포함 됨"
   2. **이스케이프 코드**\n 문자열 안에서 줄을 바꿀 때 사용  
      \t 문자열 사이에 탭 간격을 줄 때 사용  
      \\ 문자 \를 그대로 표현할 때 사용  
      \' 작은따옴표(')를 그대로 표현할 때 사용  
      \" 큰따옴표(")를 그대로 표현할 때 사용  
      \r 캐리지 리턴(줄 바꿈 문자, 현재 커서를 가장 앞으로 이동)  
      \f 폼 피드(줄 바꿈 문자, 현재 커서를 다음 줄로 이동)  
      \a 벨 소리(출력할 때 PC 스피커에서 '삑' 소리가 난다)  
      \b 백 스페이스  
      \000 널 문자  
        
      print("="\*50,"\nHello python\n","="\*50)#문자열 곱하기 및 이스케이프코드 응용  
      a = "len : 문자열 길이를 구하는 함수, 콘솔창에 len(a) 입력해보기"
   3. **문자열 인덱싱**a = "Life is too short, You need Python"  
      >>> a[3] == e  
      >>> a[-3] == h
   4. **문자열 슬라이싱**a = "python is very good!"  
      >>> a[0:5] == 'python'  
      >>> a[9:-5] == 'very'
   5. **문자열 슬라이싱 활용**  
      b = "Pithon"  
      >>> a[:1] == 'P'  
      >>> a[2:] =='thon'  
      >>> a[:1] + 'y' + a[2:] == 'Python'
   6. **문자열 포매팅**
      1. 숫자  
         >>> "I eat %d apples." % 3 == 'I eat 3 apples.'
      2. 문자열  
         >>> "I eat %s apples." % "five" == 'I eat five apples.'
      3. format을 이용한 포매팅  
         >>> number = 10  
         >>> day = "three"  
         >>> "I ate {0} apples. so I was sick for {1} days.".format(number, day) == 'I ate 10 apples. so I was sick for three days.'  
         >>>f"I ate {number} apples. so I was sick for {day} days."
   7. **문자열 관련 함수**
      1. a = 'Python is the best choice'  
         >>> a.count('i') == 2 #문자 개수 세기  
         >>> a.find('h') == 3 #문자 위치 알려주기, 문자열 미존재 시 -1 반환  
         >>> a.index('h') == 3 #문자 위치 알려주기, 문자열 미존재 시 오류를 발생  
         >>> ",".join('abcd') == 'a,b,c,d' #문자열 삽입  
         >>> a.upper() == 'PYTHON IS THE BEST CHOICE' #소문자를 대문자로  
         >>> a.lower() == 'python is the best choice' #대문자를 소문자로  
         >>> rstrip, lstrip, strip #오른쪽 공백, 왼쪽 공백, 양쪽 공백  
         >>> a.replace('Python','Programing') == 'Programing is the best choice' #문자열 바꾸기  
         >>> a.split() == ['Python', 'is', 'the', 'best', 'choice']
3. **리스트**  
   >>> a = list() == a = [] #비어있는 리스트 생성  
   >>> e = [1, 'word', ['list', 'in']] #숫자, 문자열, 리스트를 요소로 갖을 수 있음
   1. **리스트 인덱싱**a = [1, 2, ['a', 'b', ['Life', 'is']]] #삼중 리스트  
      >>> a[2][2][0] == 'Life'
   2. **리스트 슬라이싱**a = [1, 2, 3, ['a', 'b', 'c'], 4, 5] #중첩 리스트 슬라이싱 예제  
      >>> a[2:5] == [3, ['a', 'b', 'c'], 4]  
      >>> a[3][:2] == ['a', 'b']
   3. **리스트 연산**a = [1, 2, 3]  
      b = [4, 5, 6]  
      >>> a + b == [1, 2, 3, 4, 5, 6] # + 는 리스트를 합침  
      >>> a \* 3 == [1, 2, 3, 1, 2, 3, 1, 2, 3] # \* 는 리스트 반복
   4. **리스트 길이**a = [1, 2, 3, 4, 5, 6, 7]  
      >>> len(a) == 7 # len함수 기억하자
   5. **리스트 수정과 삭제**a = [1, 2, 3, 4, 5, 6, 7]  
      >>> a[3] = 6, a == [1, 2, 3, 6, 5, 6, 7] # 리스트 수정  
      >>> del a[3], a == [1, 2, 3, 5, 6, 7] # 리스트 삭제, 슬라이싱 기법 사용하여 여러 개를 한번에 삭제 가능
   6. **리스트 관련 함수**a = [1, 3, 2, 4]
      1. >>> a.append(5) == [1, 3, 2, 4, 5] #리스트에 요소 추가, append 기억하자
      2. >>> a.sort() == [1, 2, 3, 4] #리스트 정렬
      3. >>> a.reverse() == [4, 2, 3, 1] #리스트 뒤집기
      4. >>> a.index(3) == 2 #위치 반환, 리스트에 x 값이 있으면 x의 위차 값을 돌려 준다, 미존재 시 오류를 발생
      5. >>> a.insert(0, 5) == [5, 1, 3, 2, 4] #a번째에 위치에 b를 삽입
      6. >>> a.remove(3) == [1, 2, 4] #첫번쨰로 나오는 x를 삭제
      7. >>> a.pop() == 4, a == [1, 3, 2] #pop()은 리스트의 맨 마지막 요소를 추출하고 삭제
      8. >>> a.pop(1) == 3, a == [1, 2, 4] #x위치의 요소를 추출하고 삭제
      9. >>> a.count(1) == 1 #리스트 안에 x가 몇 개 있는지 찾아 반환해 줌
      10. >>> a.extend([5, 6]) == [1, 3, 2, 4, 5, 6] #리스트 + 연산과 비슷
      11. b = [5, 6]  
          >>> a.extend(b) == [1, 3, 2, 4, 5, 6] #리스트 + 연산과 비슷
4. **튜플**  
    : 튜플(tuple)은 몇 가지 점을 제외하곤 리스트와 거의 비슷하며 리스트와 다른 점은 다음과 같다.  
   - 리스트는 [ ]으로 둘러싸지만 튜플은 ( )으로 둘러싼다.  
   - 리스트는 그 값의 생성, 삭제, 수정이 가능하지만 튜플은 그 값을 바꿀 수 없다.  
   - 즉, 튜플의 항목 값은 변화가 불가능함
5. **딕셔너리**   
   : 대응 관계를 나타내는 자료형, 연관 배열(Associative array) 또는 해시(Hash)라고 한다.  
   - Key 값과 value 값 한쌍을 설정하여 Key를 통해 Value를 얻는다.  
   - Key 값에는 리스트 불가  
   - Key가 중복되었을 때 1개를 제외한 나머지 Key:Value 값이 모두 무시 됨   
     
   dic = {'name':'jh', 'phone':'01022307227', 'birth': [0, 9, 1, 6]}  
   a = {1: 'a'}  
   >>> a[2] = 'b' == {1: 'a', 2: 'b'} #딕셔너리 쌍 추가  
   >>> a['name'] = 'jh' == {1: 'a', 2: 'b', 'name': 'jh'}  
   >>> a[3] = [1, 2, 3] == {1: 'a', 2: 'b', 'name': 'jh', 3: [1, 2, 3]} #value에 리스트 추가 가능  
   >>> del a['name'] == {1: 'a', 2: 'b', 3: [1, 2, 3]} #딕셔너리 요소 삭제 key값에 해당하는 쌍이 삭제 됨
   1. **딕셔너리 관련 함수들**
      1. a = {'name':'jh', 'phone':'01022307227', 'birth': [0, 9, 1, 6]}
      2. >>> a.keys() == dict\_keys(['name', 'phone', 'birth']) #딕셔너리 a의 Key만 모아서 dict\_keys 객체를 반환
      3. >>> list(a.keys()) == ['name', 'phone', 'birth'] #Key리스트 만들기, 딕셔너리 a의 key만 모아서 리스트로 만듦
      4. >>> a.values() == dict\_values(['jh', '01022307227', [0, 9, 1, 6]]) #딕셔너리 a의 value만 모아서 dict\_values 객체를 반환
      5. >>> a.items() == dict\_items([('name':'jh'), ('phone':'01022307227'), ('birth': [0, 9, 1, 6])]) #Key와 Value의 쌍을 튜플로 묶은 값을 dict\_item 객체로 반환
      6. >>> a.clear() == {} #딕셔너리 안의 모든 요소를 삭제
      7. >>> a.get('name') == 'jh' #key로 value얻기(get), a['name']와 같음
      8. >>>> get은 요소 값이 미존재 시 None을 반환, a['unknown']은 오류를 출력
      9. >>> a.get('nonono', 'fail') == fail #딕셔너리 안에 찾으려는 Key 값이 없을 경우 미리 정해 둔 디폴트 값(fail)을 대신 가져오게 함

# **#lay the goround python\_2**

1. **집합**  
   : 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형  
   - 중복을 허용하지 않는다.  
   - 순서가 없다.  
   s1 = set([1, 2, 3, 4, 5 ,6])  
   s2 = set(4, 5, 6, 7, 8, 9])
   1. **교집합**>>> s1 & s2 == [4, 5, 6] # & 교집합  
      >>>> s1.intersection(s2) # intersection을 이용한 교집합
   2. **합집합**>>> s1 | s2 == [1, 2, 3, 4, 5, 6, 7, 8, 9] # | 합집합  
      >>>> s1.union(s2) # union을 이용한 합집합
   3. **차집합**  
      >>> s1 -s2 == [1, 2, 3] #차집합  
      >>>> s1.difference(s2) # difference를 이용한 차집합
   4. **집합 자료형 관련 함수들**
      1. s1 = set([1, 2, 3])
      2. s1.add(4) == [1, 2, 3, 4] # 값 1개 추가
      3. s1.update([4, 5, 6]) == [1, 2, 3, 4, 5, 6] # 값 여러개 추가
      4. s1.remove(2) == [1, 3] # 특정 값 제거
2. **불 자료형**: 참과 거짓을 나타내는 자료형  
   - True or False 만 가질 수 있음
   1. a = True  
      >>> type(a) == <class 'bool'>
3. **기타 함수**a = [1, 2, 3]  
   b = a
   1. **Is**>>> a is b == True # a와 b가 가르키는 객체는 동일 한가?
   2. **리스트 복사**b = a   
      >>> a[1] = 4  
      >>> a == [1, 4, 3], b == [1, 4, 3] #a와 b는 동일한 리스트를 가리키고 있기 때문에 같은 값이 출력됨  
        
      b = a[;]  
      >>> a[1] = 4  
      >>> a== [1, 4, 3], [b == [1, 2, 3] #a 변수의 값을 가져오면서 a와는 다른 주소를 가리키도록 [;] 이용해 복사  
        
      from copy import copy  
      b = copy(a)  
      >>> a[1] = 4  
      >>> a is b == False # copy 모듈을 이용 동작은 b = a[;]와 동일 하다.
   3. **변수를 만드는 여러가지 방법**a, b = ('python', 'life') || (a, b) = 'python', 'life' #튜플  
      >>> a == ('python', 'life'), b == ('python', 'life') # a, b에 같은 값을 대입할 수 있음  
      % 리스트도 가능  
      [a, b] = ['python', 'life']
   4. **여러개의 변수에 같은 값**a = b = 'python'
   5. **두 변수의 값을 교환 하기**  
      a = 5  
      b = 3  
      >>> a, b = b, a  
      >>> a == 5, b== 3 #a, b = b, a 문장을 수행한 후에는 그 값이 서로 바뀜

# **#lay the goround python\_3**

1. **제어문**  
   : 집을 지을 때 나무, 돌, 시멘트 등(자료형)은 재료가 되고, 집의 뼈대를 이루는 철근은 제어문 이다.  
   이번 장에서는 자료형을 바탕으로 제어문을 이용하여 프로그램의 구조를 만들어 보자.
   1. **If문**python은 if문에 속하는 모든 문장에 들여쓰기를 이용하여 표현

|  |
| --- |
| if <조건문>이 참이면:  수행할 문장1  수행할 문장2 elif <조건문>이 참이면 #이전 조건문이 거짓이면 elif의 <조건문> 실행  수행할 문장3  수행할 문장4  elif ...   # elif 개수 제한 없이 사용 가능 else <조건문>이 모두 거짓이면:  수행할 문장5  수행할 문장6 |

1. **조건문이란**?  
   : 참과 거짓을 판단하는 문장  
   <, >, ==, !=, >=, <= : 부등호 작다,크다,같다,같지않다,크거나같다, 작거나같다
   1. **x or y** : x와 y 둘중에 하나만 참이여도 참이다 ( +, | )
   2. **x and y** : x와 y 모두 참이여야 참이다 ( \*, & )
   3. **not x** : x가 거짓이면 참이다.
   4. **x in s** : s라는 리스트(or튜플or문자열) 안에 x가 들어있어면 참
   5. **x not in s** : s라는 리스트(or튜플or문자열) 안에 x가 들어있어면 거짓  
      # 가끔 조건먼의 참, 거짓에 따라 행동을 정의할때 아무것도 하지 않도록 설정하고 싶으면 pass를 적용
2. **조건부 표현식**if score >= 60:  
    message = "success"  
   else:  
    message = "failure"
   1. **간략화**  
      messag = "success" if score >= 60 else "failure"   
      #중요! 하이라이트! 조건부 표현식은 가독성에 유리하고 활용성이 좋음
3. **While문 #반복문**

|  |
| --- |
| coffee = 10 while True:  money = int(input("돈을 넣어 주세요: "))  if money == 300:  print("커피를 줍니다.")  coffee = coffee -1  elif money > 300:  print("거스름돈 %d를 주고 커피를 줍니다." % (money -300))  coffee = coffee -1  else:  print("돈을 다시 돌려주고 커피를 주지 않습니다.")  print("남은 커피의 양은 %d개 입니다." % coffee)  if coffee == 0:  print("커피가 다 떨어졌습니다. 판매를 중지 합니다.")  break #while문 중지 |

* 1. **while문의 맨처음으로 돌아가기**

|  |
| --- |
| #continue문을 이용 (예제로 내용 대체) a = 0 while a < 10:  a = a + 1  if a % 2 == 0: continue #continue문은 while문의 맨 처음으로 돌아가게 하는 명령어이다.  print(a) |

* 1. **무한루프**while True:  
      print("Ctrl+C를 눌러야 while문을 빠져나갈 수 있습니다.")  
     >>>Ctrl+C를 눌러야 while문을 빠져나갈 수 있습니다.  
     >>>Ctrl+C를 눌러야 while문을 빠져나갈 수 있습니다.  
     ...

1. **for문**
   1. **for문의 기본 구조**test\_list = ['one', 'two', 'three']  
      for i in test\_list:  
       print(i)  
      >>> one  
      >>> two  
      >>> three
   2. **for문 예제**

|  |
| --- |
| #marks1.py marks = [90, 25, 67, 45, 80] number = 0  for mark in marks: # 점수 리스트 marks에서 차례로 점수를 꺼내어 mark라는 변수에 대입하고 for문 안의 문장들을 수행  number = number +1   if mark >= 60:   print(f"{number}번 학생은 합격입니다.")  else:   print(f"{number}번 학생은 합격입니다.") >>> C:\doit>python marks1.py >>> 1번 학생은 합격입니다. >>> 2번 학생은 불합격입니다. >>> 3번 학생은 합격입니다. >>> 4번 학생은 불합격입니다. >>> 5번 학생은 합격입니다. # for문에서도 continue문 사용 가능하다. |

* 1. **for문과 함께 자주 사용하는 range함수**a = range(1,11) #1부터 11미만의 숫자를 포함하는 range객체를 만들어 줌  
     예제

|  |
| --- |
| #marks3.py marks = [90, 25, 67, 45, 80] for number in range(len(marks)): #len 함수는 리스트 안의 요소 개수를 반환함 즉, len(marks) 는 5가 됨  if marks[number] < 60:   continue #continue문 수행  print("%d번 학생 축하합니다. 합격입니다." % (number+1)) >>> C:\doit>python marks3.py >>> 1번 학생 축하합니다. 합격입니다. >>> 3번 학생 축하합니다. 합격입니다. >>> 5번 학생 축하합니다. 합격입니다. |

* 1. **리스트 내포 사용하기**: 리스트 안에 for문을 포함하는 리스트 내포(List comprehension)를 사용하면 좀 더 편리하고 직관적인 프로그램밍 가능
     1. 리스트 내포 사용 전  
        a = [1, 2, 3, 4]  
        result = []  
        for num in a:  
         result.append(num\*3)  
         print(result)  
        >>> [3, 6 ,9 ,12]
     2. #리스트 내포 사용 시  
        a = [1, 2, 3, 4]  
        result = [num \*3 for num in a]  
        print(result)  
        >>> [3, 6, 9, 12]
     3. 리스트 내포를 이용한 구구단 만들기  
        result = [ x\*y for x in range(2,10)  
         for y in range(1,10)]  
        print(result)  
        >>>[2, 4, 6, 8, 10, 12, 14, 16, 18, 3, 6, 9, 12, 15, 18, 21, 24, 27, 4, 8, 12, 16,20, 24, 28, 32, 36, 5, 10, 15, 20, 25, 30, 35, 40, 45, 6, 12, 18, 24, 30, 36, 42, 48, 54, 7, 14, 21, 28, 35, 42, 49, 56, 63, 8, 16, 24, 32, 40, 48, 56, 64, 72,9, 18, 27, 36, 45, 54, 63, 72, 81]
     4. 리스트내포 + 조건문  
        a = [1,2,3,4]  
        result = [num \* 3 for num in a if num % 2 == 0]   
        #num을 2로나누어서 0이면 a[num]\*3하여 result에 반환  
        print(result)  
        >>> [6, 12]

# **#lay the goround python\_4**

**프로그램 입력과 출력 -**  함수, 입력과 출력, 파일 처리 방법 등에 대해서 다룬다.  
: 반복적으로 사용되는 가치 있는 부분을 한 뭉치로 묶어서 어떤 입력값을 주었을 떄 어떤 결과 값을 돌려주기 위함

1. **함수의 구조**def 함수명(매개변수):  
    <수행할 문장1>  
    <수행할 문장2> …
   1. **예제**

|  |
| --- |
| def add(a, b):  return a+b a = 3 b = 4 c = add(a, b)  print(c) >>> 7 |

1. **매개변수와 인수**: 매개변수(parameter)매개변수는 함수에 입력으로 전달된 값을 받는 변수를 의미  
   : 인수(arguments) 수는 함수를 호출할 때 전달하는 입력값을 의미한다

|  |
| --- |
| def plus(a, b): // a, b 는 매개변수(parameter)  return a + b; |
| result = plus(1, 2); // 1, 2 는 인수(argument) printf(result); |

1. **여러 개의 입력값을 받는 함수**매개변수 기입 란에 \*args 넣어서 만든다
   1. **예제** **1**

|  |
| --- |
| def add\_many(\*args):   result = 0   for i in args:  result = result + i   return result result = add\_many(1,2,3,4,5,6,7,8,9,10) print(result) >>> 55 |

* 1. **예제 2**

|  |
| --- |
| def add\_mul(choice, \*args):   if choice == "add":   result = 0   for i in args:   result = result + i   elif choice == "mul":   result = 1   for i in args:   result = result \* i  return result   result = add\_mul('add', 1,2,3,4,5) print(result) >>> 15  result = add\_mul('mul', 1,2,3,4,5) print(result) >>> 120 |

1. **키워드 파라미터 kwargs**: 키워드 파라미터를 사용할 때는 매개변수 앞에 별 두 개(\*\*)를 붙여사용  
   : 매개변수 kwargs는 딕셔너리가 되고 모든 key=value 형태의 결괏값이 그 딕셔너리에 저장된다.
   1. **예제**1

|  |
| --- |
| def print\_kwargs(\*\*kwargs):  print(kwargs) print\_kwargs(a=1) >>> {'a': 1} print\_kwargs(name='foo', age=3) >>> {'age': 3, 'name': 'foo'} |

1. **함수의 결과값은 언제나 하나이다.**def add\_and\_mul(a,b):   
    return a+b, a\*b  
   result = add\_and\_mul(3,4)  
   >>> (7, 12) #result = (7, 12)라는 하나의 튜플 값이 반환됨
2. **2개의 결괏값처럼 받고 싶은 경우**result1, result2 = add\_and\_mul(3, 4)  
   >>> print(result1) == 7  
   >>> print(result1) == 12 #result1은 7이 되고 result2는 12가 된다.
3. **함수에 return이 2개인 경우**def add\_and\_mul(a,b):   
    return a+b # 첫 번째로 실행된 return값을 반환하여 함수가 종료 됨  
    return a\*b # 즉, 이 행은 실행되지 않는다 (없는거와 같다)
4. **Lambda**  
   : def와 동일한 역할, 보통 한줄로 간결하게 만들 떄 사용  
   : lambda 매개변수1, 매개변수2, ... : 매개변수를 이용한 표현식
   1. **예제**

|  |
| --- |
| add = lambda a, b: a+b result = add(3, 4) print(result) >>> 7  def add(a, b): return a+b result = add(3, 4) print(result) >>> 7 |

1. **사용자의 입력**
   1. **Input(“질문 내용”) #프롬프트를 띄워 질문을 하고 입력을 받음**number = input("숫자를 입력하세요: ")  
      >>> 숫자를 입력하세요: 3 # 3입력  
      >>> print(number)  
      >>> 3
   2. **print 자세히 알기**
      1. a = 123  
         print(a)  
         >>> 123
      2. a = "Python"  
         print(a)  
         >>> Python
      3. a = [1, 2, 3]  
         print(a)  
         >>>[1, 2, 3]
      4. print("life" "is" "too short") #큰따옴표(")로 둘러싸인 문자열은 + 연산과 동일하다  
         >>> lifeistoo short
      5. print("life"+"is"+"too short") #위와 실행 결과 동일 함  
         >>> lifeistoo short
      6. print("life", "is", "too short") # 문자열 띄어쓰기는 콤마로 구분 됨  
         >>> life is too short
      7. 한줄에 결괏값 출력하는 법  
         for i in range(10):  
          print(i, end=' ')  
          ...  
         >>> 0 1 2 3 4 5 6 7 8 9
2. **파일 읽고 쓰기**
   1. 파일 생성하기  
      f = open("새파일.txt", 'w')  
      f.close()
   2. 파일 열기 모드
      1. r : 읽기 모드 – 파일을 읽기만 할 때 사용
      2. w : 쓰기 모드 – 파일에 내용을 쓸 때 사용
      3. a : 추가 모드 – 파일의 마지막에 새로운 내용을 추가 시킬 때 사용
   3. **예제**

|  |
| --- |
| f = open("C:/doit/새파일.txt", 'w') for i in range(1, 11):  data = "%d번째 줄입니다.\n" % i  f.write(data) f.close() >>> C:/doit/새파일.txt 생성됨 파일 내용 >>> 1 번째 줄입니다. >>> 2 번째 줄입니다. >>> … >>> 10 번째 줄입니다. |

1. **외부에 저장된 파일을 읽는 여러 가지 방법**
   1. Readline 함수  
      : 파일의 첫 번째 줄을 읽어 출력
      1. 예제1

|  |
| --- |
| f = open("C:/doit/새파일.txt", 'r') line = f.readline() print(line) f.close() >>> 1 번쨰 줄입니다. |

* + 1. 예제 2  
       Readline 함수를 사용해 모든 행의 값을 가져 오기

|  |
| --- |
| f = open("C:/doit/새파일.txt", 'r') while True:  line = f.readline()  if not line: break  print(line) f.close() |

* 1. Readlines 함수  
     : 파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 리스트로 돌려줌

|  |
| --- |
| f = open("C:/doit/새파일.txt", 'r') lines = f.readlines() for line in lines:  print(line) f.close()  >>>print(lines)  >>> ['1행\n', '2행\n', '3행'] |

* 1. Read 함수  
     : 파일의 내용 전체를 문자열로 돌려줌

|  |
| --- |
| f = open("C:/doit/새파일.txt", 'r') data = f.read() print(data) f.close()  >>>print(data)  >>>1행  >>>2행  >>>3행 |

* 1. 파일에 새로운 내용 추가하기
     1. 파일 열기 모드를 ‘a’ 로 실행

|  |
| --- |
| f = open("C:/doit/새파일.txt",'a') for i in range(11, 20):  data = "%d번째 줄입니다.\n" % i  f.write(data) f.close() |

* 1. **with문 사용  
     :** with문을 사용하면 with 블록을 벗어나는 순간 열린 파일 객체 f가 자동으로 close되어 편리하다.

|  |
| --- |
| with open("new.txt", "a") as f:  f.write("Life is too short, you need python") |

* 1. **sys모듈로 매개변수 설정**
     1. 참조 : <https://wikidocs.net/26>

# **#lay the goround python\_5**