**■컴퓨팅 사고(computational thinking)**

: 인간이 문제를 분석하고 컴퓨터로 문제를 해결하기 위한 절차를 만드는 것 = 계산적 사고, 컴퓨팅 사고력

: 자넷 윙 교수가 2006년 컴퓨터 학술지에서 언급한 용어로 컴퓨터 교육 분야에서 상당한 이슈, 컴퓨터 사고의 기술이 단지 과학자에게만 국한되지 않고 모든 사람에게 필요한 근본적이 돼야함.

“컴퓨팅 사고는 문제의 해결책을 만드는 사고의 과정으로 이 방법으로 생성된 해결책은 컴퓨터에 의하여 효율적으로 실행될 수 있다.”

**■컴퓨팅 사고의 개념**

1) 분해(Decomposition)

복잡한 문제를 좀 더 작고 처리가능한 부분 문제로 분해

2) 패턴인식(Pattern Recognition)

부분 문제들 중에서 유사성(패턴)을 찾음

\*과거에 나타난 문제를 적용

3) 추상화(Abstraction)

중요한 정보에만 집중, 관련 없는 세부사항은 무시

\*복잡성 최소화

4) 알고리즘(Algorithms)

문제에 대한 단계적인 해결책을 개발

\*알고리즘을 프로그램으로 변환하는 과정: 프로그래밍

1->2->3->4: 컴퓨팅 사고의 과정

**■분해(Decomposition)**

: 복잡한 문제를 잘게 나누어 간단한 여러 개의 작은 문제로 분해하는 것

1) 분석(Analysis)

어떻게 복잡한 문제를 해결할 수 있는 작은 수준의 문제로 분해할 것인지에 초점을 맞히는 과정

2) 합성(Synthesis)

여러 개의 작은 문제를 모아서 주어진 문제에 대한 해답을 만드는 과정

3) 병렬처리

작은 문제를 해결하는데 다중 작업을 하는 것

\*한 주제의 작은 문제들은 병렬적이 아니라 순차적으로 해결해야한다(무조건은 아님, 동시가능시 동시!)

**■분할 정복 알고리즘(Divide and Conquer Algorithm)**

: 방대한 문제를 작은 문제로 분할하여 문제를 해결하는 알고리즘 분해기법

1) 분할(Divide)

복잡한 문제를 쉽게 풀 수 있는 유형이 비슷한 하위의 작은 문제로 분할

2) 정복(Conquer)

분할된 문제를 다시 더 작은 문제로 나누거나, 더 이상 분할되지 않는다면 작은 문제를 해결

3) 합ㄴ성(Combine)

해결한 작은 문제를 통합하여 복잡한 문제를 해결

\*이진탐색(Binary Search)

-가장 대표적인 분할정복으로 리스트에서 하나의 숫자를 찾는 알고리즘

-오름차순 또는 내림차순으로 명령되어있는 상태에서 시작

\*이진탐색 알고리즘

1) 크기 순으로 정렬된 숫자들의 리스트가 있다고 가정

2) 리스트의 중앙 값을 탐색 값과 비교(일치하면 찾은 것)

3) 중앙 값보다 작으면 탐색 값은 리스트의 전반부에 위치(리스트 후반 부는 탐색 값에서 제외, 탐색 값이 클 경우 반대임)

4) 1~3 반복

**■패턴인식(Pattern Recognition)**

-패턴

: 여러가지 문제들이 공유하는 유사성 또는 특성이다

-패턴인식

: 복잡한 문제를 효율적으로 해결할 수 있도록 작게 분해된 문제들 사이의 유사성 또는 패턴을 찾는 과정

-일반화

: 패턴을 발견하게 되면 패턴에 근거하여 추측할 수 있고 또 규칙을 만들 수 있으며 패턴을 이용하여 일반적인 문제들을 해결 할 수 있다.

(문제에서 패턴을 더 많이 찾을수록 일반적으로 문제해결이 더 쉽고 빠르게 진행된다.)

■**추상화(Abstraction)**

-추상

: 문제를 해결하기 위하여 불필요한 세부사항을 제거하는 기법

: 복잡한 시스템의 구체적인 예로부터 공통적인 특성을 추려내서 일반적인 개념을 형성하는 과정

-추상화하는 방법

: 추상화하고자하는 대상의 특징, 특성을 찾는다. 불필요한 세부사항을 지운다.

**■알고리즘(Algorithm)**

: 문제를 해결하기 위한 단계적 절차, 주어진 문제의 해결방법을 사람이나 컴퓨터가 이해할 수 있는 명령어로 정확하게 기술한 것, 특정한 일을 수행하는 명령어들의 집합

#조건

-입력: 외부에서 제공되는 0개 이상의 입력이 존재

-출력: 1개 이상의 출력 존재

-명백성: 명령어의 의미는 명확해야 함

-유한성: 한정된 수의 명령어 실행 후 반드시 종료

-유효성: 각 명령어는 실행가능한 연산이어야 함

#기술방법-1

-순서도(Flowchart)

: 알고리즘 도형을 이용하고 시각적으로 표현한 것

**<사용되는 기호>**

-화살표: 알고리즘이 진행되는 방향

-모서리가 둥근 사각형: 수행의 시작, 종료

-직사각형: 처리

-마름모: 판단 (예/아니오 질문 해당, 이 도형에서는 두개의 화살표가 나옴)

-기울어진 사각형: 입력이나 출력

**<제어구조>**

-순차구조(Sequence): 명령어들이 순차적으로 실행되는 구조

-선택구조(Selection): 둘 중 하나의 명령어를 선택하여 실행되는 구조

-반복구조(Iteration): 동일한 명령어가 반복되면서 실행되는 구조

#기술방법-2

-의사코드(Pseudo-code)

-START – END

-IF -> THEN(참)

->ELSE(거짓)

-WHILE: 반복(참이면 진행 거짓이면 멈춤)

**■파이썬(Python)**

: 귀도 반 로섬(네델란드 수학자)이 개발한 인터프린트 언어

\*인터프린트 언어: 한 줄씩 소스코드를 해석해서 그 때마다 실행해 결과를 바로 알 수 있다.

+컴파일러 언어: 소스코드를 실행가능한 기계어로 일괄 번역 후에 번역이 완료된 언어를 실행(C언어)-인터프린트 언어보다 실행속도가 빠르다

**■파이썬의 특징**

-파이썬은 인간다운 언어이다

-파이썬은 문법이 쉬워 빠르게 배울 수 있다

-파이썬은 무료이지만 강력하다

-파이썬은 간결하다 (가독성이 좋다)

-파이썬은 프로그래밍을 즐기게 해준다

-파이썬은 개발속도가 빠르다

\*파이썬으로 할 수 없는 일: 대단히 빠른 속도를 요구하거나 하드웨어를 직접 건드려야 하는 프로그램

**■파이썬 기초문법**

-파이썬은 대소문자를 구분한다.

-사칙연산

더하기: +

나누기: /

곱하기: \*

-대입연산자: = (‘a는 b이다.’)

\*파이썬에서 ‘같냐’라는 연산자는 ‘==’이다(not=)

-조건문 if

-반복문 for, while

-함수

-들여쓰기: Tab키, Space 4번 (혼용불가)

-주석처리: #또는 ‘’,”” 3개 사용

-출력: F5키(전체), F9키(부분)

-콘솔: 저장X, 스크립트창: 저장O

■**숫자형(Number)**

: 숫자 형태로 이루어진 자료형

-정수형(Integer)

123, -345, 0 …

-실수형(Floating-point)

소수점이 포함된 숫자

(일반적) 1.2, -3.45 …

(컴퓨터식 지수표현) 4.24E10, 4.24e-10

4.24E10 = 4.24 \*10^10

4.24e-10 = 4.24\*10^-10

\*E나 e 중 어느 것을 사용해도 무방!

-8진수, 16진수(사용 잘 안 함)

8진수 = 0o 또는 0O (숫자 0 + 알파벳 o)

16진수 = 0x

-사칙연산

X\*\*Y = X^Y: \*\* 연산자는 제곱을 뜻

7%3 = 1: %연산자는 나눗셈 후 나머지 뜻

7//4 = 1: //연산자는 나눗셈 후 몫을 뜻

\*곱하기, 나누기 연산자 보다 제곱연산자의 우선순위가 더 높다. 순서 주의. 필요에 따라 괄호 사용

\*수치 값이 클 때 ‘,’를 쓰지 않고, ‘\_’를 쓴다

1천만: 10\_000\_000

\*퍼센트는 소수로 바꾼다(%기호는 연산자이므로)

90% = 0.9

**■문자열(String)**

: 단어 등으로 구성된 문자들의 집합

(따옴표로 둘러싸여 있으면 모두 문자열)

**■문자열 만드는 방법**

1) 큰 따옴표/작은 따옴표로 양쪽 둘러싸기

2) 큰 따옴표/작은 따옴표 3개를 연속으로 써서 양쪽 둘러 싸기

\*한 쪽은 큰 따옴표, 다른 쪽은 작은 따옴표 = error

**■문자열 안에 작은 따옴표나 큰 따옴표 포함시킬 때**

1) 문자열 안에 작은 따옴표 포함시키기

>>>food = “Python’s favorite food is perl”

ㄴ큰 따옴표로 둘러싸야 한다

\*>>>food = ‘Python’s favorite food is perl’

‘Python’이 문자열로 인식

구문오류(SyntaxError) 발생(SyntaxError:invalid syntax)

2) 문자열 안에 큰 따옴표(“) 포함시키기

>>>say = ‘”Python is easy.” He says.’

ㄴ작은 따옴표로 둘러싸야 한다

3)문자열 안에 큰 따옴표와 작은 따옴표 포함시키기

>>>food = ‘Python＼’s favorite food is perl’

>>>say = “＼”Python is very easy. ＼” He says.”

ㄴ문자로 쓰이는 “,’ 앞에 백슬래시를 사용한다.

\*백슬래시 키보드 위치 = enter 키 위 \

**■여러 줄인 문자를 변수에 대입하고 싶을 때**

1) 줄을 바꾸는 이스케이프 코드 ‘＼n’ 삽입하기

2) 연속된 작은 따옴표, 큰 따옴표 3개 사용하기

>>>multiline =”””Life is too short

Life is too short

You need Python”””

**■문자열 연산하기**

1) 더해서 연결하기

>>>head = ‘Python’

>>>tail = ‘is fun!’

>>>head + tail

‘Python is fun!’

2)곱하기

>>>a = ‘Python’

>>>a \* 2 #두 번 반복하라는 뜻

‘PythonPython’

3)문자열 길이 구하기

>>>a = ‘Life is too short”

>>>len(a) #len()함수 사용

17 #공백도 포함함

**■문자열 인덱싱**

인덱싱(indexing) = 가리킨다 #문자의 위치를 지정함

“파이썬은 0부터 숫자를 센다”

>>>Life is too short = a

>>>a[3]

‘e’

a[n] : a문자열의 n번째 요소 추출

a[-1]은 뒤에서 첫번째 a[0]=a[-0]

인덱싱에서는 한가지 요소만 추출 가능하다, 단어를 추출하기 위해서는 슬라이싱이 필요

**■문자열 슬라이싱**

-슬라이싱(Slicing): 잘라낸다 – 원하는 단어 추출

>>>a = ‘Life is too short’ #에서 Life 추출

1)인덱싱

>>>b = a[0]+a[1]+a[2]+a[3]

>>>b

‘Life’

2)슬라이싱

>>>a[0:4] #a문자열에서 0~3인덱스 추출

‘Life’

[start:end]=start~(end-1)까지 추출

a[n:] = n번째~끝

a[:n] = 처음 ~ n-1번째

a[:] = 처음~끝

\*n이 음수일 수도 있다

a[start:end:step]

start~end 까지 step 만큼 건너뛰면서 추출

>>>a = ”Life is too short”

>>>a = [2:11:2]

‘f sto’

**■문자열 수정하기**

문자열 자료형은 요솟값을 변경할 수 없다.

-슬라이싱 활용하여 수정

>>>a = ‘Python’

>>>a[:1] + a[2:]

‘Python’

**■문자열 포매팅**

: 문자열 안의 특정한 값을 바꾸거나 삽입하는 기법

1) 숫자 바로 대입

>>>”I eat %d apples.” %3

‘I eat 3 apples.’

정수형 숫자인 경우 넣고 싶은 자리에 ‘%d’ 적고 삽입할 숫자는 가장 뒤 % 뒤에 넣는다

2) 문자열 바로 대입

>>>”I eat %s apples.” %”five”

‘I eat five apples.’

문자열을 삽입하고 싶은 경우 넣고 싶은 자리에 ‘%s’ 적고 가장 뒤 %에 문자열을 넣는다

\*\*문자열을 대입할 때는 % 뒤에 큰 따옴표나 작은 따옴표를 써줘야 한다.

3) 숫자 값을 나타내는 변수로 대입

>>>number = 3

>>>”I eat %d apples.” %number

‘I eat 3 apples.’

4) 2개 이상 값 넣기

>>>number = 10

>>>day =”three”

>>>”I ate %d apples. So I was sick for %s days.” %(number, day)

2개 이상의 값을 넣으려면 마지막 % 다음 괄호 안에 콤마(,)로 구분하여 각각의 값을 넣는다

1. 지정자 개수 2. 순서 3. 코드 자료형

\*단, %문자열 포맷 코드와 괄호 안의 자료형의 순서 일치!

**<문자열 포맷 코드>**

%s: 문자열

- %s 코드는 어떤 형태는 error X(정수여도 상관없음)

%d: 정수

%f: 부동소수(실수형)

%%: literal % (%문자 자체)

->>>”Error is %d%%” %98

‘Error is 98%’ ##%% 안 하면 값오류(Value Error)

%c: 문자 1개

%o: 8진수

%x: 16진수

**■포맷 코드와 숫자 사용하기**

1) 정렬과 공백

>>>”%10s” %”hi”

‘ hi’

->%10s는 전체 길이가 10인 문자열 공간에서 대입되는 값을 오른쪽 정렬하고 나머지는 공백!

>>>”%-10sjane” %’hi’

‘hi jane’

-> -는 왼쪽 정렬, jane은 10칸에 속하지 않음

2) 소수점 표현하기

>>>”%0.4f” %3.42134234

‘3.4213’

-> ’.’은 소수점 포인트이고 그 뒤 4는 소수점 뒤에 나올 숫자의 개수이다. (반올림 적용!)

>>>”%10.4f” %3.42134234

‘ 3.4213’ #소수점 ‘.’도 하나의 칸에 속함

**■format 함수를 이용한 포매팅**

포맷 코드와 다르게 함수를 이용하면 문자형, 정수형, 실수형을 지정할 필요가 없다!

1) 숫자 바로 대입

>>>”I eat {0} apples.”.format(3)

‘I eat 3 apples.’

2) 문자열 바로 대입

>>>”I eat {0} apples.”.format(“five”)

‘I eat five apples.’

3) 숫자 값 변수 대입

>>>number = 3

>>>”I eat {0} apples.”.format(number)

‘I eat 3 apples.’

4) 2개 이상의 값 넣기

>>>number = 3

>>>day = “three”

>>>”I ate {0} apples. So I was sick for {1} days.”.format(number, day)

\*.format(a,b) = a 다음에 b가 들어간다

\*{n} n이 꼭 0,1,2,3,4…차례일 필요는 없다

\*하나의 값을 두 군데 이상 넣을 수 있다

5) 이름으로 넣기

>>>”I ate {number} apples. So I was sick for {day} days.”.format(number = 10, day = 3)

name = value와 같은 형태가 있어야 한다.

6) 인덱스와 이름 혼용

>>>”I ate {0} apples. So I was sick for {day} days.”.format(10, day = 3)

7) 왼쪽 정렬

>>>”{0:<10}”.format(“hi”)

‘hi ‘

\*꺾새 방향 왼쪽=왼쪽 정렬

8) 오른쪽 정렬

>>>”{0:>10}”.format(“hi”)

‘ hi’

9) 가운데 정렬

>>>”{0:^10}”.format(“hi”)

‘ hi ‘

10) 공백 채우기

>>>”{0:=^10}.format(“hi”)

‘====hi====’

\*공백 대신 대체할 문자일 경우 한 글자만 가능, 문자여도 “”로 묶지 않는다

11) 소수점 표현하기

>>>y=3.42134234

>>>”{0:10.4f}”.format(y)

‘ 3.4213’

\*”{0:4.4f}”.format(y)여도 오류X, 공백 없이 출력

\*>>>”{0:010.4f}”.format(y) #공백을 0으로 채움

‘00003.4213’

12) {문자} 출력

>>>”{{and}}”.format() #문자그대로 중괄호 사용

‘{and}’

**■ f 문자열 포매팅**

1) 표현식

>>>age = 30

>>>f’나는 내년이면 {age+1}살이 된다.’

‘나는 내년이면 31살이 된다.’

\*f문자열은 표현식을 지원한다(+,-,…)

2) 딕셔너리

>>>d= {‘name’:‘홍길동’, ‘age:30’}

>>>f’내이름{d[“name”]}, 나이{d[“age”]}살’

‘내 이름 홍길동, 나이 30살’

3) 정렬

>>>f’{“hi”:<10}’

‘hi ‘

>>>f’{“hi”:>10}’

‘ hi’

>>>f’{“hi”:^10}’

‘ hi ‘

4) 공백 채우기

>>>f’{“hi”:=^10}’

‘====hi====’

5) 소수점

>>>y=3.41234234

>>>f’{y:10.4f}’

‘ 3.4123’

‘

6) {}문자

>>>f’{{and}}’

‘{and}’

**■ 문자열 관련 함수**

꼴 = 문자열. 함수명()

1) 문자열 개수 세기(count)

>>>a = “hobby”

>>>a.count(‘b’)

2

2) 위치 알려주기1(find)

>>>a = “Python is the best choice”

>>>a.find(‘b’)

14

>>>a.find(‘k’)

-1 #k가 없다 > ‘-1’ 반환

3) 위치 알려주기2(index)

>>>a=”Life is too short”

>>>a.index(‘t’)

8

>>>a.index(‘k’)

ValueError (Find 함수와의 차이점!)

4) 문자열 삽입(join)

‘삽입할 문자’.join(문자열)

>>>”,”.join(abcd) #각각의 문자 사이에 ‘,’ 삽입

‘a,b,c,d’

5)소>대문자 (upper)

>>>a = ”hi”

>>>a.upper()

‘HI’

6) 대>소문자 (lower)

7) 왼쪽 공백 지우기(lstrip)

8) 오른쪽 공백 지우기(rstrip)

9) 양쪽 공백 지우기(strip)

10)문자열 바꾸기(replace)

>>>a = “Life is too short”

>>>a.replace(“Life”, “your leg”)

‘your leg is too short”

11)문자열 나누기(split)

>>>a.split() #공백을 기준으로 나눔

[‘Life’, ‘is’, ‘too’, ‘short’] #리스트에 저장됨

**■ 리스트**

: 여러 종류의 데이터를 한 곳에 저장하고 창조할 수 있는 데이터 구조

: 리스트를 만들 때에는 대괄호 []로 묶어주고 각 요소 값은 쉼표로 구분한다.

a = []

b = [1,2,3]

c = [1,2,’Life’,’is’]

d = [1,2,[‘Life’,’is’]] #리스트 안에 리스트 가능

\*비어 있는 리스트는 a=list()로 생성할 수도 있다

: 리스트 안에는 어떤 자료형도 포함시킬 수 있다.

**■리스트 인덱싱**

>>>a = [1,2,3]

>>>a[0]

1

>>>a[0] + a[2]

4

\*a[-1]=마지막 리스트

<중첩된 리스트>

a=[1,2,3,[‘a’,’b’,’c’]]

>>>a[3]

[‘a’,’b’,’c’]

>>>a[-1][0] #a의 [-1]번째 인덱스 중 [0]번째

‘a’

\*\*삼중, 사중 등 여러 번 중첩될 수도 있다.

**■리스트 슬라이싱**

>>>a=[1,2,3,4,5]

>>>b=a[:2] #처음부터 a[1]까지

>>>c=a[2:] #a[2]부터 마지막까지

>>>b

[1,2]

>>>c

[3,4,5]

>>>a[0:2]

[1,2]

**■리스트 연산하기**

1)리스트 더하기: +

2)리스트 반복하기: \*

3)리스트 길이 구하기 (len())

>>>a=[1,2,3]

>>>len(a)

3 #인덱스 숫자가 아닌 그대로의 길이

\*리스트 연산 오류

>>>a = [1,2,3]

>>>a[2]+”hi”

TypeError: unsupported operand type(s) for +: ‘int’ and ‘str’

->정수형과 문자열은 서로 더할 수 없으므로 형 오류 발생

->숫자 3을 문자 3으로 전환

>>>str(a[2]) + “hi”

3hi #str()함수는 정수나 실수를 문자열로 바꿈

\*형 확인 함수: type()함수 #데이터의 자료형을 반환

\*int: 정수형, float: 실수형, str: 문자열

■리스트의 수정과 삭제

-수정

>>>a = [1,2,3]

>>>a[2] = 4

>>>a

[1,2,4]

-삭제 (del함수)

>>>a = [1,2,3]

>>>del a[1]

>>>a

[1,3]

>>>a=[1,2,3,4,5]

>>>del a[2:] #슬라이싱 사용

>>>a

[1,2]

**■리스트 관련 함수**

1) 요소추가 (append)

>>>a =[1,2,3]

>>>a.append(4)

>>>a

[1,2,3,4] #마지막에 추가된다

2) 정렬 (sort)

>>>a = [1,4,3,2]

>>>a = sort()

>>>a

[1,2,3,4] #숫자: 오름차순, 문자: 알파벳순

3) 뒤집기 (reverse)

>>>a = [‘a’, ’c’, ‘b’]

>>>a.reverse()

>>>a

[‘b’,’c’,’a’] #a.sort(reverse = True) : 내림차순 정렬

4) 위치 반환 (index)

>>>a=[1,2,3]

>>>a.index(3)

2 #a문자열의 3의 위치를 반환함

>>>a.index(1)

0 #만약 찾고자 하는 값이 없으면 에러

>>>a=[1,2,3,1,2,3]

>>>a.index(3)

2 #가장 먼저 나오는 위치를 반환함

>>>a.index(3,3)

5 #index(x,y) 함수는 y번째 이후의 x값 반환

5) 요소 삽입(insert)

>>>a = [1,2,3]

>>>a.insert(0,4) #a[0]위치에 4 삽입

[4,1,2,3]

6) 요소 제거(remove)

>>>a = [1,2,3,1,2,3]

>>>a.remove(3) #3은 요소 값(not 위치)

[1,2,1,2,3]

#’첫번째’로 나오는 값을 삭제한다

(한번 더 실행하면 다시 삭제)

7) 요소 끄집어내기(pop)

>>>a = [1,2,3]

>>>a.pop(1) #리스트의 맨 마지막 요소를 반환하고

3 그 요소는 삭제한다

>>>a

[1,2]

>>>a = [1,2,3]

>>>a.pop(1) #1은 인덱스 값(위치)-remove랑 차이점

2

>>>a

[1,3]

8) 요소 개수 세기 (count)

>>>a = [1,2,3,1]

>>>a.count(1) #1은 요소 값!

2

9) 확장(extend)

>>>a = [1,2,3]

>>>a.extend([4,5])

>>>a

[1,2,3,4,5]

\*extend(x)의 x에는 ‘리스트’만 올 수 있으며 원래의 a 리스트에 x리스트를 더한다.

\*리스트 안에 리스트가 들어가는 것이 아니다

\*a.extend([4,5])는 a+=[4,5]와 동일하다

\*+=는 계산한 값을 원래 값에 할당한다고 해서 할당 연산자라고 한다. x+=1은 x=x+1이다

**■튜플 자료형**

: 리스트와 대체로 비슷하지만, 몇 가지는 다름

-리스트: [] / 튜플: ()

-리스트: 값의 생성, 수정, 삭제 기능 / 튜플 값 변경X

>>>t1 = ()

>>>t2 = (1,)

>>>t3 = (1,2,3)

>>>t4 =1,2,3

>>>t5 = (‘a’,’b’,(‘ab’,’cd’))

#1개의 요소만을 가질 때는 반드시 요소 뒤 콤마

#t4처럼 괄호 생략 무방

#프로그래밍이 실행되는 동안 값이 바뀌지 않기를 바랄 때 튜플 사용

**■튜플 변경하기 (요소 값 변경)**

>>>t1 = (1,2,’a’,’b’)

>>>del t1[0]

TypeError: ‘tuple’ object does not support item assignment

■튜플 다루기

1) 인덱싱

>>>t1 = (1,2,’a’,’b’)

>>>t1[0]

1

>>>t1[3]

‘b’

2) 슬라이싱

>>>t1 = (1,2,’a’,’b’)

>>>t1[1:]

(2.’a’,’b’)

3) 튜플 더하기

>>>t2 = (3,4)

>>>t1+t2

(1,2,’a’,’b’,3,4)

4) 튜플 곱하기

>>>t2 \* 3

(3,4,3,4,3,4,)

5) 튜플 길이구하기

>>>t1 = (1,2,’a’,’b’)

>>>len(t1)

4

**■딕셔너리**

: 대응 관계를 나타낼 수 있는 자료형

Key : Value의 형태, 연관배열/해시라고 부름

-리스트나 튜플처럼 순차적으로 해당 요소 값을 구하지 않고 key를 통해 value를 얻음

{Key1:Value1, Key2:Value2, Key3:Value3,…}

\*각 쌍들은 {}로 둘러싸여 있다.

\*쉼표로 구분한다

Key: 변하지 않는 값/Value: 변하거나 변하지 않는 값

>>>dic: {‘name’ : ’pey, ‘phone’ : ‘01102’, ‘birth’ : ‘1122’}

>>>a = {1 : ’hi’} #key로 정수 값, Value에 문자열

>>>a = {‘a’:[1,2,3]} #Value에 리스트 사용가능

**■딕셔너리 쌍 추가하기**

>>>a = {1: ‘a’}

>>>a[2] = ‘b’ #{2 : ’b’} 쌍 추가

>>>a

{1: ‘a’, 2: ‘b’}

>>>a[‘name’] = ‘pey’ #{‘name’ : ‘pey’}쌍 추가

>>>a

{1: ‘a’, 2: ‘b’, ‘name’: ‘pey’}

>>>a[3] = [1,2,3]

>>>a

{1: ‘a’, 2: ‘b’, ‘name’: ‘pey’, 3: [1,2,3]}

**■딕셔너리 요소 삭제하기**

>>>del a[1] #key가 1인 key: value쌍 삭제

>>>a

{‘name’: ‘pey’, 3: [1,2,3], 2: ‘b’}#딕셔너리와 순서는 무관

**■딕셔너리에서 Key 사용해 Value 얻기**

>>>grade = {‘pey’:10, ‘julliet’:99}

10 #Ke가 ‘pey’인 딕셔너리의 Value를 반환

>>>grade[‘julliet’]

99 #Key가 ‘julliet’인 딕셔너리의 Value를 반환

>>>a = {1:’a’, 2:’b’}

>>>a[1] #1은 인덱스가 아닌 key를 뜻함

‘a’

\*a[Key]로 입력하여Key에 해당하는 Value얻는다

**■딕셔너리 만들 때 주의사항**

1) Key는 고유한 값이므로 중복되는 Key 값을 설정해 놓으면 하나를 제외한 나머지가 무시된다

>>>a = {1:’a’, 1: ‘b’} #1이라는 key값이 중복사용

>>>a

{1: ‘b’} #1:’a’쌍이 무시된

2) Key에 리스트는 쓸 수 없다. (튜플은 가능)

: Key에는 변하지 않는 값을 써야하기 때문

대신, 리스트가 Value에 사용될 수는 있다

>>>a = {[1,2] : ‘hi’} #리스트를 키로 사용

TypeError: unhashable type: ‘list’

**■ Key 리스트 만들기(keys)**

>>>a = {‘name’ : ‘pey’, ‘birth’ : ‘1122’}

>>>a.keys()

dict\_keys([‘name’, ‘phone’, ‘birth’])

a.keys()는 a의 Key만을 모아서 dict\_keys객체를 반환

#type()으로 확인하면 아직 리스트는 아님

#리스트 고유의 append, insert, pop등의 함수는 수행 불가

>>>for k in a.keys(): #for은 반복, k는 변수

… Print(k)

…

name #첫번째 반복

birth #두번쨰 반복

dict\_keys 객체를 리스트로 변환시키려면

>>>list(a.keys()) #list()함수를 사용

[‘name’, ‘birth’]

**■ Value 리스트 만들기**

>>>a.values() #values()함수 사용

dict\_values([‘pey’, ‘1122’])

**■ Key, Value 쌍 얻기(items)**

>>>a.items()

dict\_items([(‘name’, ‘pey’), (‘birth’, ‘1122’)])

items 함수는 Key와 Value의 쌍을 튜플로 묶은 값을 dict\_items객체로 돌려준다

**■ Key: Value 쌍 모두 지우기(clear)**

>>>a.clear()

>>>a

{}

**■ Key로 Value 얻기(get)**

>>>a={‘name’ : ‘pey’, ‘birth’ : ‘1122’}

>>>a.get(‘name’)

‘pey’

>>>a.get(‘birth’)

‘1122’

a.get(‘name’)은 a[‘name’]을 사용했을 때와 동일한 결과값을 받는다

다만, a[‘nokey’]처럼 존재하지 않는 키로 값을 가져오려고 할 경우 a[‘nokey’]는 Key 오류를 발생시키고 a.get(‘nokey’)는 None을 돌려준다는 차이가 있다.

>>>a.get(‘foo’, ‘bar’)

‘bar’

딕셔너리 안에 찾으려는 Key 값이 없고, 미리 정해 둔 디폴트 값을 대신 가져오게 하고 싶을 때

get.(x, ‘디폴트 값’)을 사용

**■ 해당 키가 딕셔너리 안에 있는지 조사하기(in)**

>>>a = {‘name’ : ’pey’, ‘birth’ : ’1122’}

>>>’name’ in a #a 딕셔너리 안에 ‘name’키가 있냐

True

>>>’email’ in a

False

**■ 집합 자료형**

집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형

중괄호 {}를 감싸서 표현하지만 딕셔너리와는 다르게 key, value 값이 없음

Set 키워드를 이용해 만듦

>>>s1 = set([1,2,3])

>>>s1

{1,2,3}

>>>s2 = set(“Hello”)

>>>s2

{‘e’,’H’,’l’,’o’} #l은 한 개만 나온다. 중복허용x

s1 = {1,2,3}으로 정수형을 바로 만들 수 있다

하지만, set(1,2,3)으로 하면 error이 나온다. (중괄호 때문에) 소괄호 일 경우 set(1)은 가능하다. (요소 값이 하나이기 때문에)

>>>s4 = set(1,2,3,4)

>>>s4

TypeError: set expected at most 1 argument, got 4

**■ 집합 자료형의 특징**

-중복을 허용하지 않는다

-순서가 없다 -> 인덱싱을 지원하지 않는다

\*set 자료형에 저장된 값을 인덱싱으로 접근하려면 리스트나 튜플로 변환해야한다.

>>>s1 = set([1,2,3])

>>>l1 = list(s1) #리스트로 변환

>>>l1

[1,2,3]

>>>l1[0]

1

>>>t1 = tuple(s1) #튜플로 변환

>>>t1

(1,2,3)

>>>t1[0]

1

\*변환시키지 않으면

TypeError: ‘set’ object is not subscriptable

**■교집합, 합집합, 차집합 구하기**

>>>s1 = set([1,2,3,4,5,6])

>>>s2 = set([4,5,6,7,8,9])

1) 교집합

>>>s1 & s2

{4,5,6}

>>>s1.intersection(s2)

{4,5,6}

2) 합집합

>>> s1 | s2 #shift + 역슬래시 키

{1,2,3,4,5,6,7,8,9}

>>>s1.union(s2)

{1,2,3,4,5,6,7,8,9}

3) 차집합

>>>s1 - s2

{1,2,3}

>>>s2 - s1

{8,9,7} #집합 자료형은 순서가 없다

>>>s1.difference(s2)

{1,2,3}

>>>s2.difference(s1)

{8,9,7}

**■ 집합 자료형 관련 함수**

1) 값 1개 추가하기(add)

>>>s1 = set([1,2,3])

>>>s1.add(4)

>>>s1

{1,2,3,4}

2) 값 여러 개 추가하기(update)

>>>s1 = set([1,2,3])

>>>s1.update([4,5,6])

>>>s1

{1,2,3,4,5,6} #중복 허용 안함

3) 특정 값 제거하기(remove)

>>>s1.remove(2)

>>>s1

{1,3}

**■ 불 자료형**

: 참(True)과 거짓(False)을 나타내는 자료형

\*True나 False는 파이썬의 예약어로 첫 문자를 항상 대문자로 사용해야 한다.

>>>1 == 1 #1과 1이 같은가?

True

>>>2 >1

True

>>>2<1

False

|  |  |  |
| --- | --- | --- |
| 자료형 | 값 | 참 / 거짓 |
| 문자열 | “Python” | 참 |
| “” | 거짓 |
| 리스트 | [1,2,3] | 참 |
| [] | 거짓 |
| 튜플 | () | 거짓 |
| 딕셔너리 | {} | 거짓 |
| 숫자형 | 0이 아닌 숫자 | 참 |
|  | 0 | 거짓 |
|  | None | 거짓 |

>>>bool([1,2,3])

True

>>>bool([])

False

>>>bool(0)

False

>>>bool(3)

True

**■ 변수**

: 객체를 가리키는 것

변수 이름 = 변수에 저장할 값

\*파이썬은 변수에 저장된 값을 스스로 판단하여 자료형을 지정

>>>a = 1

1의 값의 정수형 자료형(객체)이 자동으로 메모리에 생성됨. 변수 a는 1이 저장된 메모리 구조

=>a는 1을 저장하고 있다.

>>>id(a)

4202020400 #컴퓨터 메모리의 주소 값

**■ 변수를 만드는 방법**

>>>a ,b=(‘python’, ‘life’) #튜플(튜플은 괄호 생략가능)

>>>[a,b] = [‘python’, ‘life’] #리스트

>>>a = b = ‘python’

\*\*오른쪽부터 순서이다. 1. b=python, 2. a=python

>>>a = 3

>>>b = 5

>>>a, b= b, a #a와 b의 값을 바꿈

>>>a

5

>>>b

3

**■ 리스트의 복사**

>>>a = [1,2,3]

>>>b = a #b와 a는 완전히 동일하다.

>>>id(a)

4321234321

>>>id(b)

4321234321

주소 값까지 동일하기 때문에 a를 수정하면 b도 수정된다

>>>a is b

True

**<다른 주소를 가리키며 복사하기>**

1) [:] 사용

리스트 전체를 가리키는 [:]을 사용해서 복사

>>>a = [1,2,3]

b = a[:]

a[1] = 4

>>>a

[1,4,3]

>>>b

[1,2,3]

2) copy 모듈 사용

>>>from copy import copy #copy모듈 불러내다

>>>a = [1,2,3]

>>>b = copy(a) #a[:]와 동일

>>>b is a

False

<<기말고사 범위>>

■ **if문**

조건을 판단하여 해당 조건에 맞는 상황을 수행하는 데 쓰는 것

if 조건문: #참

수행할 문장1

수행할 문장2

else: #거짓, else문에는 조건문이 필요없다

수행할 문장3

수행할 문장4

\* 콜론 안 찍으면 error 발생, else문은 if문 없이 독립적으로 사용할 수 없다. (else문 자체는 생략가능 but, else문 사용시 if문 반드시 필요)

■ **들여쓰기**

if문을 만들 때는 조건문 바래 아래 문장부터 모든 문장에 들여쓰기를 해주어야 한다. (space, tab혼용X)

\*\*Error발생 위치 잘 보기

money = True

if money:

print(“택시를”)

print(“타고”) #들여쓰기 X, **여기선 Error X**

print(“가라”) #여기서 Error 발생

money = True

if money:

print(“택시를”)

print(“타고”)

print(“가라”) #여기서 Error 발생

money = True

if money:

print(“택시를”)

print(“타고”) #여기서 Error 발생

print(“가라”)

\*\*에러 위치 (IndentationError 들여쓰기 에러)

1. 들여쓰기가 안 되어있는 곳 ‘다음 줄’부터 에러발생

2. 들여쓰기가 두 번 되어있는 곳부터 에러발생

■ **연산자**

|  |  |
| --- | --- |
| 비교 연산자 | 설명 |
| x<y | x가 y보다 크다 |
| x>y | x가 y보다 작다 |
| x==y | x와 y가 같다 |
| x!=y | x와 y가 같지 않다 |
| x>=y | x가 y보다 크거나 같다 |
| x<=y | x가 y보다 작거나 같다 |

>>>x = 3

>>>y = 2

>>>x==y

False

>>>x!=y

True

>>>money = 2000

>>>if money >=3000:

print(“택시를 타고 가라”)

>>>else:

print(“걸어가라”)

걸어가라 #따옴표 안 붙음!!

|  |  |
| --- | --- |
| 연산자 | 설명 |
| x or y | x와 y 둘 중에 하나만 참이면 참이다 |
| x and y | x와 y 모두 참이어야 참이다 |
| not x | x가 거짓이면 참이다 |

>>>money = 2000

>>>card = True

>>>if money >=3000 or card:

print(“택시를 타고 가라”)

else:

print(“걸어가라”)

택시를 타고 가라

■

|  |  |
| --- | --- |
| in | not in |
| x in 리스트 | x not in 리스트 |
| x in 튜플 | x not in 튜플 |
| x in 문자열 | x not in 문자열 |

>>>1 in [1,2,3] #1이 [1,2,3] 안에 있는가

True

>>>1 not in [1,2,3]

False

>>>’j’ not in ‘python’

True

>>>pocket = [‘paper’, ‘cellphone’, ‘money’]

>>>if ‘money’ in pocket:

print(“택시를 타고 가라”)

else:

print(“걸어가라”)

택시를 타고 가라

**■pass**

조건문의 참, 거짓에 따라 실행할 행동을 정의할 때, 아무런 일도 하지 않도록 설정

<주머니에 돈이 있으면 가만히 있고, 주머니에 돈이 없으면 카드를 꺼내라>

>>>pocket = [‘paper’, ‘money’, ‘cellphone’]

>>>if ‘money’ in pocket:

pass

>>>else:

print(“카드를 꺼내라”)

#pocket리스트 안에 money 문자열이 있기 때문에 pass가 수행되고 아무 결괏값도 안뜬다

■ **elif**

else + if (else다음에 if문이 올 때 elif문으로 사용가능)

>>>pocket = [‘paper’, ‘cellphone’]

>>>card = True

>>>if ‘money’ in pocket:

print(“택시를 타고 가라”)

elif card: #주머니에 돈이 없고 카드가 있으면

print(“택시를 타고 가라”)

else:

print(“걸어 가라”)

택시를 타고 가라

\*if, elif, else를 모두 사용할 때 기본 구조

if 조건문:

수행문장

elif 조건문2:

수행문장2

elif 조건문3:

수행문장3

else:

\*elif는 개수에 제한 없이 사용될 수 있다.

■ **if문 한 줄 작성**

if문 다음 수행할 문장을 콜론 뒤에 바로 적는다.

>>>pocket = [‘paper’, ‘money’, ‘cellphone’]

>>>if ‘money’ in pocket: pass

else: print(“카드를 꺼내라”)

■ **조건부 표현식**

조건문이 참인 경우 if 조건문 조건문이 거짓인 경우

<일반 표현식>

if score>=60:

message = “success”

else:

message = “failure”

<조건부 표현식>

message = “success” if score >=60 else “failure”

**■ while문**

조건문이 참인 동안에 while문 아래의 문장이 반복해서 수행됨

while 조건문:

수행할 문장1

수행할 문장2 …. #수행문장은 if문이 될 수도 있다

>>>treetHit = 0

>>>while treeHit < 10:

treeHit = treeHit +1

print(“나무를 %d번 찍었습니다.” %treeHit)

if treeHit ==10:

print(“나무 넘어갑니다.”)

나무를 1번 찍었습니다.

나무를 2번 찍었습니다.

…

나무를 10번 찍었습니다.

나무 넘어갑니다.

■ while문 만들기

>>>prompt = “”” #문자열 입력

1. Add

2. Del

3. List

4. Quit

Enter number:”””

>>>number = 0

>>>while number !=4: #입력 번호가 4가 아니면 반복

print(prompt)

number = int(input())

1. Add

2. Del

3. List

4. Quit

Enter number: #4를 입력하면 조건문이 거짓이 되어 while문을 빠져나가게 된다. (창에는 아무것도 안뜸)

■ while문 강제로 빠져나가기

break문 사용

coffee = 10

while True:

money = int(input(“돈을 넣어주세요: “))

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. 판매중지 합니다.”)

**break**

**■** while문의 맨 처음으로 돌아가기

continue문 사용

>>>a = 0 #0을 1로 바꾸면 짝수 출력 예제

>>>while a < 10:

a = a +1

if a %2 ==0: continue

print(a)

1

3

5

7

9 #홀수만 출력하는 예제, 짝수는 다시 a로 돌아감

# a를 2로 나누었을 때 나머지가 0이면 맨 처음으로 돌아간다

■ **무한 루프**

while문으로 구현

<기본 구조>

while True: #while문의 조건문이 true이므로 항상 참

수행할 문장1

수행할 문장2

….

>>>while True:

print(“Ctrl + C를 눌러야 while문을 빠져나갈 수 있습니다.”)

Ctrl + C를 눌러야 while문을 빠져나갈 수 있습니다.

Ctrl + C를 눌러야 while문을 빠져나갈 수 있습니다.

…

\*파이썬에서는 Ctrl+C를 눌러야 나갈 수 있다.

\*Spyder에서는 Ctrl + .(dot)을 사용하면 커널(console)을 다시 시작하거나 프로그램이 중지된다

■ **random 함수**

import random

random.random() #0.0~1.0 사이의 실수 반환(1.0포함X)

random.uniform(a,b) #a~b실수 반환(a,b 포함)

random.randint(a,b) #a~b 정수 반환(a,b 포함)

random.randrange(a,b) #a~b 정수 반환(b포함X)

random.range(b) #0~b 사의 정수 반환(b포함X)

random.choice(seq) #문자열, 튜플, 리스트, range 사이에서 무작위로 한 개 원소 포함

random.sample(seq 또는 set, N) #시퀀스 타입 또는 set 타입에서 랜덤하게 N개의 수 만큼 인자를 반환,

#unique, 순서에 상관없이 인자를 리스트로 반환

random.shuffle(seq) #데이터의 순서를 무작위로 바꿔줌, 리스트만 가능

**■ for 문**

반복문

<기본구조>

for 변수 in 리스트(또는 튜플, 문자열)

수행할 문장1

수행할 문장2

….

>>>test\_list = [‘one’, ‘two’, ‘three’]

>>>for i in test\_list: #리스트의 길이 = 반복횟수

print(i)

one

two

three

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

>>>for (first, last) in a:

print(first + last)

3 #first = 1, last = 2

7

11

<60점 이상 합격 - 2번째 방법(1번째는 교재)>

marks = [90,25,67,45,80]

number = 0

for mark in marks:

number = number +1

if mark < 60: continue

print(“%d번 학생 축하합니다. 합격입니다.”%number)

\*for문에도 continue문을 사용하면 for문의 처음으로 돌아가게 된다.

**■ range함수**

>>>a = range(10)

>>>a

range(0,10) #0,1,2,3,4,5,6,7,8,9

range(10)은 0부터 10 미만의 숫자를 포함하는 range객체를 만들어 준다.

range(시작 숫자, 끝 숫자) #끝 숫자 range미포함

>>>add = 0

>>>for i in range(1,11): #1~10숫자 대입

add = add+i

>>>print(add)

55 #1~10까지 더해짐

**<60점 이상이면 합격 - 3번째 방법>**

marks = [90,25,67,45,80]

for number in range(len(marks)):

if marks[number] < 60 : continue

print(“%d번 학생 축하합니다. 합격입니다.%(number+1))

**<for과 range를 활용한 구구단>**

>>>for i in range(2,10):

for j in range(1,10):

print(i\*j, end = “”)

print(‘’)

2 4 6 8 10 12 14 16 18

3 6 9 12 15 18 21 24 27

……

**■리스트 내포**

리스트 안에 for문을 포함

[표현식 for 항목 in 반복 가능 객체 if 조건]

>>> a = [1,2,3,4]

>>>result = [num \* 3 for num in a if num %2 == 0]

>>>print(result) #리스트 숫자 중 짝수에만 3을 곱함

[6,12]

**■ 함수**

파이썬 함수의 구조

def 함수이름(매개변수): #매개변수가 없어도 괄호 존재

수행할 문장1

수행할 문장2

…

return 결괏값 #결괏값은 오직 return명령어로만 받음

>>>def add(a,b): #함수 선언(함수를 만든 것)

return a+b #a,b는 매개변수

>>>a = 3 #함수 호출

>>>b = 4 #3,4는인수

>>>c = add(a,b) #위에 만든 함수 꼴

>>>print(c)

7

**■ 매개변수와 인수**

매개변수 = 함수에 입력으로 전달된 값을 받는 변수

인수 = 함수를 호출할 때 전달하는 입력 값

**<일반적인 함수>**

>>>def add(a,b):

result = a+b

return result

>>>a = add(3,4)

>>>print(a) #a라 두어도 실행됨

7

**<입력값이 없는 함수>**

>>>def say():

return ‘Hi’

#say라는 이름의 함수를 만들없지만, 매개변수 부분을 나타내는 함수 이름 뒤의 괄호가 비어있다.

>>> a = say()

>>>print(a)

Hi #따옴표 없다

위 함수를 쓰기 위해서는 say()처럼 괄호 안에 아무 값도 넣지 않아야 한다.

**결괏값을 받을 변수 = 함수이름()**

**<결괏값이 없는 함수>**

>>>def add(a,b):

print(“%d, %d의 합은 %d입니다.”%(a,b,a+b))

>>>add(3,4)

3,4의 합은 7입니다.

#문장이 출력되었으나 이건 print문의 수행할 문장에 해당되는 것이지, 결괏값은 없다. 결괏값은 오직 return명령어로만 돌려받을 수 있다.

>>>a = add(3,4)

3, 4의 합은 7입니다.

>>>print(a)

None #return에 의해 결괏값을 받지 못함.

#None은 거짓을 나타내는 자료형

함수이름(입력인수1, 입력인수2, …)

**<입력값도 결괏값도 없는 함수>**

>>>def say():

print(‘Hi’)

>>>say()

Hi

함수이름()

**<매개변수 지정하여 호출>**

>>>def add(a,b):

return a+b

>>>result = add(a=3, b=7)

>>>print(result)

10

순서에 상관없이 사용할 수 있다

>>>result = add(b=5, a=3) #b=5, a=3이 됨.

>>>print(result)

8

**<입력값이 몇 개가 될지 모를 때>**

def 함수이름(\*매개변수)

수행할 문장

…

>>>def add\_many(\*args): #args는 임의로 정한 이름

result = 0

for i in args:

result = result +i

return result

#add\_many 함수는 입력값이 몇 개이든 상관이 없다.

\*args처럼 매개변수 이름 앞에 \*를 붙이면 입력값을 전부 모아서 튜플로 만들어 주기 때문이다.

>>>result = add\_many(1,2,3)

>>>print(result)

6

>>>result = add\_many(1,2,3,4,5,6,7,8,9,10)

>>>print(result)

55

------

>>>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

**■ 키워드 파라미터**

키워드 파라미터를 사용할 때는 매개변수 앞에 별 두 개를 붙인다.

>>>def print\_kwargs(\*\*kwargs):

print(kwargs)

>>>print\_kwargs(a=1)

{‘a’ : 1}

>>>print\_kwargs(name = ‘foo’, age = 3)

{‘age’ : 3, ‘name’ : ‘foo’}

입력값이 모두 딕셔너리로 만들어져서 출력된다. 즉 \*\*kwargs처럼 매개변수 이름 앞에 \*\*을 붙이면 매개변수 kwargs는 딕셔너리가 되고 모든 key = value형태의 결괏값이 그 딕셔너리에 저장된다.

**■ 함수의 결괏값은 언제나 하나이다**

>>>def add\_and\_mul(a,b):

return a+b, a\*b

>>>result = add\_and\_mul(3,4)

>>>result

(7,12) #튜플 값 생성, 오류 안남

만약 하나의 튜플 값을 2개의 결괏값처럼 받고 싶다면

>>>result1, result2 = add\_and\_mul(3,4)

처럼 호출하며 된다. result1,result2=(7,12)가 되어 result1은 7이 되고 result2는 12가 된다.

----

>>>def add\_and\_mul(a,b):

return a+b

return a\*b #실행되지 않을 것

>>>result = add\_and\_mul(2,3)

>>>print(result)

5

#return문을 만나는 순간 결괏값을 돌려주고 다음 함수를 빠져나간다.

**■ return의 또 다른 쓰임새**

특별한 상황일 때 함수를 빠져나가고 싶다면 return을 단독으로 써서 즉시 빠져나갈 수 있다.

>>>def say\_nick(nick):

if nick == “바보”:

return

print(“나의 별명은 %s입니다.”%nick)

#’별명’을 입력으로 전달받아 출력하는 함수, 이 함수 역시 결괏값은 없다. 문자열을 추출하는 것 뿐 결괏값은 return’문’에 의해서만 생성된다.

>>>say\_nick(‘야호’)

나의 별명은 야호입니다.

>>>say\_nick(‘바보’)

>>>

**■매개변수에 초깃값 미리 설정하기**

>>>def say\_myself(name, old, man=True):

print(“나의 이름은 %s입니다.”%name)

print(“나이는 %d살입니다.”%old)

if man:

print(“남자입니다.”)

else:

print(“여자입니다.”)

>>>say\_myself(“박응용”, 27) #man에 입력값 없지만 초깃값 True를 갖게되어 ‘남자입니다.’를 출력함

>>>say\_myself(“박응용”, 27, True)

나의 이름은 박응용입니다.

나이는 27살입니다.

남자입니다.

\*\*함수의 매개변수에 초깃값을 설정할 때는 위치에 주의해야 한다.

만약, def say\_myself(name, man=True, old):

라고 설정하고 say\_myself(“박응용”, 27)을 호출하면

SyntaxError: non-default argrument follows default argument 라는 오류 메시지가 뜬다. 초깃값을 설정해 놓은 매개변수 뒤에 초깃값을 설정해 놓지 않은 매개변수는 사용할 수 없다는 뜻이다. 따라서 초기화시키고 싶은 매개변수를 항상 뒤에 둬야한다.

\*def add(a, b=0, c=0)

result = a +b+c

return result

print(add(1,5)

6

**■ 함수 안에서 선언한 변수의 효력 범위**

a = 1

def vartest(a):

a = a + 1

vartest(a)

print(a)

vartest 함수에서 매개변수 a의 값에 1을 더했으니까 2가 출력될 것 같지만, 결괏값은 1이 나온다. 그 이유는 함수 안에서 새로 만든 매개변수는 함수 안에서만 사용하는 ‘함수만의 변수’이기 때문이다. 즉, def vartest(a)의 매개변수 a는 함수 안에서만 사용하는 변수이고 밖의 변수 a가 아니다.

def vartest(a):

a = a + 1

vartest(3)

print(a)

print(a)문장은 오류가 난다. print(a)에서 입력받아야 하는 a변수를 찾을 수 없기 때문이다. (a가 4로 변함)

**■ 함수 안에서 함수 밖의 변수를 변경하는 방법**

**1. return 사용**

a = 1

def vartest(a):

a = a+1

return a

a = vartest(a)

print(a)

**2. grobal 명령어 사용하기**

a = 1

def vartest():

global a

a = a + 1

vartest()

print(a)

**■ lambda**

보통 함수를 한 줄로 간결하게 만들 때 사용, def랑 동일

def를 사용할 정도로 복잡하지 않거나, def를 사용할 수 없는 곳에 사용

>>>add = lambda a, b: a+b

>>>result = add(3,4)

>>>print(result)

7

\*lambda함수는 새 변수를 만들 수가 없음. (변수 없이 한 줄씩으로 표현할 수 있어야 함)

\*lambda예약어로 만든 함수는 return명령어가 없어도 결과값을 돌려줌

**■ 사용자 입력**

>>>a = input()

Life is too short, you need python #직접 입력

>>>a

‘Life is too short, you need python’

\*input은 입력되는 모든 것을 문자열로 취급

\*숫자도 문자열로 인식되어 사칙연산 불가능, 숫자로 쓰고 싶으면 int()함수 사용하여 형변환

>>>number = input(“숫자를 입력하세요:”)

숫자를 입력하세요: 3 #직접입력, 따옴표 없음

>>>print(number)

3

**■ print문**

>>>a =123 ; >>>print(a)

123

>>>a = ‘Python’ ; >>>print(a)

Python #따옴표 안 붙음

>>>a = [1,2,3] ; >>>print(a)

[1,2,3]

**\*큰따옴표로 둘러싸인 문자열은 + 연산과 동일**

>>>print(“life””is””too short”)

>>>print(“life”+”is”+”too short”)

lifeistoo short

**\*문자열 띄어쓰기는 콤마로**

>>>print(“life”, ”is”, ”too short”)

life is too short

**\*한줄에 결괏값 사용하기 - 매개변수 end활용**

>>>for i in range(10):

print(i, end=’’) #두 줄 띄우고 싶으면 역/n

**■ 파일 생성하기**

파일객체 = open(파일이름, 파일열기모드)

|  |  |
| --- | --- |
| 파일열기모드 | 설명 |
| r | 읽기 모드 - 파일을 읽기만 할 때 |
| w | 쓰기 모드 - 파일에 내용을 쓸 때 |
| a | 추가 모드 - 파일의 마지막에 새로운 내용을 추가할 때 |

f = open(“C:/doit/새파일.txt”.’w’)

f.close()

#모드 반드시 ‘’로 감싸야한다

#밑줄 부분은 절대경로

#f.close생략 가능하지만, 다시 사용시 오류나므로 반드시 닫는다.

**\*현재 디렉토리 확인하기**

<프롬포트 창>

import os

print(os.getcwd())

<콘솔 창>

>>>pwd #작업중인 파일의 이름을 뜻함

**■ 파일을 쓰기모드로 열어 출력값 적기**

f = open(“C:/doit/새파일.txt”, ‘w’)

for i in range(1,11): #1~10까지 대입

data = “%d번째 줄입니다.n” %i

f.write(data)

f.close()

#화면 출력이 아니므로 화면에는 안 뜸, 해당 파일에 들어가야 결과가 뜸

<모니터에 띄우기>

for i in range(1,11):

data = “%d번째 줄입니다. 역/n” %i

print(data)

**■ 외부 파일 읽는 방법**

**<readline 함수 사용>**

f = open(“C:/doit/새파일.txt”, ‘r’)

line = f.readline()

print(line)

f.close

readline()함수는 파일의 맨 첫 번째 줄을 읽어 출력

모든 줄을 읽고 싶을 때는

f = open(“C:/doit/새파일.txt”, ‘r’)

while True:

line = f.readline()

if not line: break

print(line)

f.close

while True: 무한루프 안에서 f.realine()을 사용해 파일을 계속 읽는다. 더 이상 읽을 줄이 없으면 break를 수행-None을 출력한다.

**<readlines함수 사용>**

f = open(“C:/doit/새파일.txt”, ‘r’)

lines = f.readline**s**()

for line in lines:

print(line) #line이랑 lines 구분!

f.close

readlines함수는 파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 **리스트**로 돌려줌

[“1번째 줄입니다.”, “2번째 줄입니다.”,…,”10번째 줄입니다”] 붙어서 나오므로 한 줄만 띄우고 싶을 때는 print(line, end = ‘’)로 써야한다.

**<read함수 사용>**

f = open(“C:/doit/새파일.txt”, ‘r’)

data = f.read()

print(data)

f.close()

‘1 line /n2 line /n3 line …./n10 line/n’ #/=역/

f.read()는 파일의 내용 전체를 문자열로 돌려준다.

**■ 파일에 새로운 내용 추가하기**

f = open(“C:/doit/새파일.txt”, ‘a’)

for i in range(11,20): #11~19

data = “%d번째 줄입니다./n” %i

f.write(data)

f.close()

추가모드로 파일을 열었기 때문에 파일이 원래 가지고 있던 내용 바로 다음부터 결괏값을 적기 시작한다.

**■ with문과 함께 사용하기**

f = open(“foo.txt”, ‘w’)

f.write(“Life is too short, you need python”)

f.close()

with open(“foo.txt”, ‘w’) as f:

f.write(“Life is too short, you need python”)

with문을 사용하면 with블록을 벗어나는 순간 열린 파일 객체 f가 자동으로 close되어 따로 close안해도 됨.

**■ 클래스; 설계도**

1. 객체지향프로그래밍(OOP, Object Oriented Programming)

: 프로그램 구현에 필요한 객체를 파악하고 각각의 객체들의 역할이 무엇인지를 정의하여 객체들 간의 상호작용을 통해 프로그램을 만드는 것

2. 객체: 물리적으로 존재하거나, 추상적인 것 중에서 자신의 속성과 동작을 가지는 모든 것

클래스라는 틀에서 생겨난 실체, 객체는 속성(필드)과 메서드(동작)로 구성

클래스로 만든 객체는 고유, 독립적성격을 가짐

3. 대표적인 언어: C++, C#, Java등이 있음

4. OOP기본 구성요소: 클래스, 객체, 매서드, 필드

class Calculator: #클래스 명의 첫글자는 대문자

def \_\_init\_\_(self):

self.result = 0

def add(self, num):

self.result += num

return self.result

cal1 = Calculator()

cal2 = Calculator()

print(cal1.add(3))

print(cal1.add(4))

print(cal2.add(3))

print(cal2.add(7))

3

7

3

10

**■ 사칙연산 클래스 만들기 (틀)**

>>>class FourCal:

pass

>>>a = FourCal()

>>>type(a)

<class ‘\_\_main.FourCal’> #객체a타입은 FourCal클래스

>>>a.setdata(4, 2) #객체명.메소드()

-----

>>>class FourCal:

def setdata(self, first, second):

self.first = first

self.second = second

\*메서드의 첫 번째 매개변수 self를 명시적으로 구현하는 것은 파이썬만의 독특한 특징이다. 자바 같은 언어는 첫 번째 매개변수 self가 필요없다.

setdata 매서드 호출

>>>a = FourCal()

>>>a.setdata(4,2)

self.first = 4

self.second = 2

a.first = 4

a.second = 2

>>>a = FourCal)\_

>>>a.setdata(4,2)

>>>print(a.first)

4

>>>print(a.second)

2

>>>a = FourCal()

>>>b = FourCal()

>>>a.setdata(4,2)

>>>print(a.first)

4

**■ 메서드의 또 다른 호출 방법**

>>>a = FourCal()

>>>FourCal.setdata(a,4,2)

‘클래스 이름.메서드’형태로 호출할 때는 객체 a를 첫 번째 매개변수 self에 꼭 전달해야 한다.

**■ 사칙연산 클래스(더하기 기능)**

>>>class FourCal:

def setdata(self, first, second):

self.first = first

self.second = second

def add(self):

result = self.first + self.second

return result

>>> a= FourCal()

>>>a.setdata(4,2)

>>>print(a.add())

6

**■사칙연산 클래스 모든 기능**

>>>class FourCal:

def setdata(self, first, second):

self.first = first

self.second = second

def add(self):

result = self.first + self.second

return result

def mul(self):

result = self.first \* self.second

return result

def sub(self):

result = self.first - self.second

return result

def div(self):

result = self.first / self.second

return result

**■ 동작 확인**

>>>a = FourCal()

>>>b = FourCal()

>>>a.setdata(4,2)

>>>b.setdata(3,8)

>>>a.add()

6

>>>a.mul()

8

>>>b.add

11

>>>b.div()

0.375

**■ 생성자**

객체가 생성될 때 자동으로 호출되는 메서드

\_\_init\_\_를 사용하면 이 메서드는 생성자가 된다.

>>>class FourCal:

def \_\_init\_\_(self, first, second):

self.first = first

self.second = second

\*setdata가 따로 팔요 없다.

\*(self, first, second)로 되어있으면 객체 2개가 필요하단 뜻이다. 만약, self만 있으면 객체 호출 때 a = FourCal()만 해도 된다.

>>> a = FourCal()

TypeError: \_\_init\_\_() missing 2 required positional arguments : ‘first’ and ‘second’

>>>a = FourCal(4,2)

>>>a.add()

6

>>>a.div()

2.0

**■ 클래스의 상속**

상속은 기존클래스는 그대로 놔둔 채 클래스의 기능을 확장시킬 때 주로 사용한다.

class 클래스 이름(상속할 클래스 이름)

>>>class MoreFourCal(FourCal)

def pow(self): #거듭제곱 메서드

result = self.first \*\* self.second

return result

>>>a = MoreFourCal(4,2)

>>>a.pow

16

**■ 메서드 오버라이딩**

>>>a = FourCal(4,0)

>>>a.div()

ZeroDivisionError: division by zero

0으로 나누면 오류가 뜨는데 오류가 아닌 0을 돌려주고 싶을 때 오버라이딩을 한다.

>>>class SafeFourCal(FourCal):

def div(self):

if self.second == 0:

return 0

else:

return self.first / self.second

>>>a = SafeFourCal(4, 0) #자식클래스 호출

>>>a.div()

0

**■ 클래스 변수**

클래스 이름.클래스 변수

>>>class Family:

lastname = “김”

>>>a = Family()

>>>b = Family()

>>>print(a.lastname)

김

>>>print(b.lastname)

김

>>>Family.lastname = “박”

>>>print(a.lastname)

박

>>>print(b.lastname)

박

클래스 변수는 클래스로 만든 모든 객체에 공유됨

\*\*

>>>a.lastname = “김” #객체변수로 저장됨

>>>print(a.lastname)

김

>>>print(b.lastname)

박 #객체변수가 없음(클래스 값으로 출력)

**■ 모듈**

함수나 변수 또는 클래스를 모아 놓은 파일

def add(a,b):

return a+b

def sub(a,b):

return a-b

모듈을 불러와 사용할 때에는 같은 디렉터리에 저장되어야 한다.(콘솔창에 pwd치고 확인)

>>>import mod1

>>>print(mod1.add(3,4))

7

>>>print(mod1.sub(4,2))

2

1. import 모듈이름

2. import 모듈이름 as 이름 (문자열 표시 ‘’ 필요 X)

ㄴ모듈이름이 너무 길 때 간단한 이름으로 대체

\*사용자 정의 모듈(파이썬에서 기본으로 제공하는 표준모듈은 바로 import써서 사용가능)

**<함수 이름만 쓰고 싶은 경우>**

from 모듈 이름 import 모듈 함수

이 형식을 사용하면 모듈이름을 붙이지 않고 바로 해당 모듈의 함수를 쓸 수 있다.

>>>from mod1 import add

>>>add(3,4) #함수 이름만

7

다만, mod1 파일의 add함수만 사용가능

add함수와 sub함수 둘 다 사용하고 싶을 때는

1. from mod1 import add, sub

2. from mod1 import\*

# \*표시는 모든 함수를 불러서 사용하겠다는 뜻이다.

**■ if\_\_name\_\_==”\_\_main\_\_”:**

def add(a, b):

return a+b

def sub(a, b):

return a-b

if\_\_name\_\_==”\_\_main\_\_”:

print(add(1,4))

print(sub(4,2))

직접 파일을 실행했을 때는 참이되어 if문 다음 문장 수행, 대화형 인터프리터나 다른 파일에서 이 모듈을 사용할 때는 거짓이 되어 if 다음 문장 수행X

\*\*\_\_name\_\_변수

현재 모듈의 이름을 담고 있는 내장 변수

직접 실행된 모듈의 경우 \_\_main\_\_값을 가지게 됨. 직접 실행되지 않은 import된 모듈은 모듈의 이름(파일명)을 가지게 됨

**원의 넓이 계산 모듈**

PI = 3.141592

class Math:

def solv(self, r):

return PI\*(r\*\*2)

def add(a, b):

return a + b

>>>a = mod2.Math()

>>>print(a.solv(2)) #반지름이 2인 원의 넓이

12.566368

**■ 파일이 존재하는지 확인하는 방법**

1. os.path.exists(‘파일명’ 또는 ‘폴더명’)

2. os.path.isfile(‘파일명’) #파일만 확인

3. os.path.isdir(‘폴더명’) #디렉토리만 확인

->True(있)또는 False(없) 반환

\*(“./DoitPython/inFile1.txt”)에서 .(dot)은 현재 디렉토리를 나타냄-상대경로에서 나타남

\*(C:/Users/~/)는 절대경로

**■ 오류 발생 경우**

1. 디렉터리 안에 없는 파일을 열려고 시도했을 때

FileNotFoundError 오류

2. 0으로 숫자를 나누는 경우

ZeroDivisionError

3. 리스트에서 없는 값일 경우

>>>a=[1,2,3]

>>>a[4]

IndexError

**■ 오류 예외 처리 기법**

**<try, except문>**

try:

…

except[발생오류[as 오류메시지 변수]]:

----

try:

a=[1,2]

print(a[3])

4/0

except:

print(“오류발생!!”)

오류발생!!

----

**<발생 오류와 오류 메시지 변수까지 포함>**

try:

a=[1,2]

print(a[3])

4/0

except ZeroDivisionError as e:

print(e)

except IndexError as e:

print(e)

list index out of range

division by zero #print(a[0])처럼 첫 문장 맞을 때 출력

변수명e에 데러에 대한 메시지 저장

에러 메시지를 같이 출력

**<try, finally문>**

f = open(‘foo.txt’, ‘w’)

try:

#무언가를 수행한다.

finally:

f.close()

foo.txt 파일을 쓰기 모드로 연 후에 try문을 수행한 후 예외 발생 여부와 상관없이 finally 절에서 f.close()로 열린 파일을 닫을 수 있다.

**■ 여러 개의 오류**

try:

a =[1,2]

print(a[3])

4/0

except ZeroDivisionError:

print(“0으로 나눌 수 없습니다.”)

except IndexError:

print(“인덱싱 할 수 없습니다.”)

인덱싱 할 수 없습니다.

인덱싱 오류가 먼저 발생하고 zerodivision오류는 발생하지 않는다.

try:

a=[1,2]

print(a[3])

4/0

except ZeroDivisioError as e:

print(e)

except IndexError as e:

print(e)

list index out of range 오류메시지 출력

**<2개 이상 오류 동시 처리>**

try:

a=[1,2]

print(a[3])

4/0

except (ZeroDivisionError, IndexError) as e:

print(e)

괄호를 사용하여 함께 묶어 처리

**<try, except, else, finally문>**

try:

except:

else:

finally:

try 문에서 오류가 발생하면 except문 실행

오류가 발생하지 않으면 else문이 실행

finally문은 오류가 발생하든 않든 무조건 실행

a,b = 4.0

try:

a/b

except:

print(‘오류발생’)

else:

print(f’{a}/{b} = {a/b:.2f}’)

finally:

print(‘finally 영역은 무조건 실행’)

오류발생

finally 영역은 무조건 실행

(b가 1일 때)

4/1 = 4.00

finally 영역은 무조건 실행

**■ 예외 종류**

|  |  |
| --- | --- |
| 종류 | 설명 |
| ImportError | import문에서 오류 |
| IndexError | 리스트 등 첨자의 범위 벗어남 |
| KeyError | 딕셔너리에서 key없음 |
| KeyboardInterrupt | 프로그램 실행 중 Ctrl+C 누를 때 |
| NameError | 변수명이 없는 것에 접근할 때 |
| RecursionError | 재귀호출의 횟수가 시스템 설정보다 넘칠 때(1000번) |
| RuntimeError | 실행이 발생할 때 |
| SyntaxError | 문법상 오류가 발생할 때 |
| TypeError | 변수형의 오류가 발생할 때 예)문자열-문자열 연산 |
| ValueError | 함수의 매개변수에 잘못된 값을 넘길 때 예)int(‘파이썬’) |
| ZeroDivisionError | 0으로 나눌 때 |
| IOerror | 파일 처리 등 오류일 때 |

**■ 내장 함수**

파이썬에 이미 있는 함수, 외부모듈과 달리 import가 필요하지 않음.

**abs** : 숫자 입력 받았을 때 절댓값을 돌려주는 함수

>>>abs(-1.2)

1.2

**all** : 반복가능한 자료형 x를 입력 인수로 받으며 이 x가 모두 참이면 True, 거짓이 하나라도 있으면 False

>>>all([1,2,3,0])

False #리스트에서 요소 0은 거짓

#all[]=True 빈 값은 참

**any** : 하나라도 참이 있으면 True 모두 거짓일 때 False

>>>any([1,2,3,0])

True

>>>any([0, “”])

False

**dir** : 객체가 자체적으로 가지고 있는 변수나 함수를 보여줌

>>>dir([1,2,3])

[‘append’, ‘count’, ‘extend’ …]

#리스트가 사용할 수 있는 함수

**divmod** : 2개의 숫자를 입력받아 a를b로 나눈 몫과 나머지를 튜플 형태로 돌려줌

>>>div(7,3)

(2,1)

**enumerate** : 순서가 있는 자료형을 입력받아 반복되는 구간에서 객체가 어느 위치에 있는지 알려줌

>>>for i, name in enumerate([‘body’, ‘foo’, ‘bar’]):

print(i, name)

0 body

1 foo

2 bar

**eval** : 실행가능한 문자열을 입력받아 문자열을 실행

>>>eval(‘1+2’)

3

>>>eval(‘divmod(4,3)’)

(1,1)

>>>eval(“‘hi’+’a’”)

‘hia’ #사용 자제

**filter** : 반환 값이 참인 것만 묶어서 돌려줌

def positive(l): #filter함수 없을 때

result = []

for i in l:

if i > 0

result.append(i)

return result

print(positive([1,-3,2,0,-5,6]))

[1,2,6]

-----

def positive(x): #filter함수 사용

return x>0

print(list(filter(positive,[1, ,-3,2,0,-5,6])))

[1,2,6]

------

람다함수랑 같이 사용가능

>>>list(filter(lambda x : x>0, [1, ,-3,2,0,-5,6]))

[1,2,6]

**list** : 반복가능한 자료형s를 입력받아 리스트로 만듦

>>>list(“python”)

[‘p’, ‘y’, ‘t’, ‘h’, ‘o’, ‘n’]

**map** : map(f, iterable) 입력받은 자료형의 요소와 f함수 결과를 묶어서 돌려줌

def two\_times(numberList): #map함수 사용 안함

result = []

for number in numberList:

result.append(number\*2)

return result

result = two\_times([1,2,3,4])

print(result)

[2,4,6,8]

>>>def two\_times(x): return\*2

>>>list(map(two\_times, [1,2,3,4,]))

[2,4,6,8]

-----

lambda사용

>>>list(map(lambda a:a\*2 [1,2,3,4,]))

[2,4,6,8]

**max** : 반복가능한 자료형을 입력 받아 최댓값 돌려줌

>>>max([1,2,3])

3

>>>max(“python”)

‘y’ #아스키코드값 중 가장 큰 값

**min** : 반복가능한 자료형을 입력 받아 최솟값 돌려줌

>>>max([1,2,3])

1

>>>max(“python”)

‘h’ #아스키코드값 중 가장 작은 값

**pow** : pow(x,y) x의 제곱한 결괏값 돌려줌

>>>pow(2,4)

16

**round**: 숫자를 입력받아 반올림

>>>round(4.6)

5

>>>round(5.678, 2) #소수점 2자리까지 반올림

5.68

\*올림, 내림은 파이썬 내장함수에서 지원x

->math모듈을 이용하여 거기서 제공하는 ceil/floor, trunc사용

\*\***ceil** : 올림(내장함수 아님)

>>>import math

>>>math.ceil(-3.14)

-3

\*\***floor, trunc** : 내림

>>>import math

>>>math.floor(-3.14)

-4

>>>math.trunc(-3.14) \*int와 같은 결과 반환(정수형)

-3

**sorted** : 입력값을 정렬한 후 결과를 리스트로 돌려줌

>>>sorted((3,1,2))

[1,2,3]

\*리스트 자료형에도 sort함수가 있지만 리스트 자료형의 sort함수는 리스트 객체 그 자체를 정렬만 할 뿐 저장하여 결과를 돌려주진 않는다.

**str** : 문자열 형태로 객체 변환

>>>str(3)

‘3’

>>>str(‘hi’.upper())

‘HI’

**sum** : 입력받은 리스트나 튜플의 모든 요소 합

>>>sum([1,2,3])

6

>>>sum([4,5,6])

15

tuple : 반복가능한 자료형을 입려받아 튜플로 변환

>>>tuple(“abc”)

(‘a’,’b’,’c’)

>>>tuple([1,2,3])

(1,2,3)

**zip** : 동일한 개수로 이루어진 자료형을 묶어줌

>>>list(zip([1,2,3],[4,5,6]))

[(1,4),(2,5),(3,6)]

>>>list(zip(“abc”, “def”))

[(‘a’,’d’),(‘b’,’e’),(‘c’,’f’)]