<점프 투 파이썬 정리>

오예슬

기본
-pwd #현재 작업하고 있는 디렉토리 path
-변수 이름 = 변수에 저장할 값 (객체, 자료형)

A
abs() #절댓값 돌려줌
aiter()
all() #반복 가능한 자료형의 모든 자료가 참 True, 거짓이 하나라도 있으면 False를 돌려줌
any()
anext()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
del 자료형 #삭제(슬라이싱 이용시 한번에 삭제 가능)
delattr()
dict()
dir() #객체가 자체적으로 갖고 있는 변수/함수 보여줌
divmod()

E
enumerate() #순서가 있는 자료형(리스트,튜플,문자열)을 마치 딕셔너리의 key와 value처럼 index와 value값으로 돌려줌
  for idx, i in enumarate("hello"):
    idx = 0 1 2 3 4 ...
      i = 'h' 'e' 'l' 'l' 'o;
eval()
exec()

F
filter(함수이름, 반복가능한 자료형) #자료형을 함수에 입력했을 때 참인 것만 걸러서 돌려줌
float() #문자열을 실수로
format() #문자열 안에 어떤 값을 삽입할 때
frozenset()

G
getattr()
globals() #함수안에서 함수밖의 변수 직접 사용

H
hasattr()
hash()
help() #함수에 대해 자세히 알려주는 함수
hex()

I
id() #객체의 주소를 돌려줌
input() #사용자 입력을 받을 때 사용
int() #문자열을 숫자(정수)로
isinstance()
issubclass()
iter()

L
len(자료형) #길이 알려줌(띄어쓰기 포함)
list() #비어있는 리스트 생성
list(s) #반복 가능한 자료형s를 입력받아 리스트로 변환
locals()

M
map(함수,반복가능자료형) #자료형의 각요소를 함수에 입력한 결과를 돌려줌
max(반복자능한자료형) #최댓값을 돌려줌
memoryview()
min(반복가능한자료형) #최솟값을 돌려줌

N
next()

O
object()
oct()
open(파일이름,읽기방법) #파일 객체를 불러옴.
ord()

P
pow() #power의 줄임. 제곱한 결과값 반환.
print() #출력
property()

R
range(,)　　#주어진 범위의 연속된 숫자
repr()
reversed()
round()

S
set()　#집합
setattr()
slice()
sorted() #반복가능 자료형을 정렬한 후 리스트로 돌려줌
sorted(,reverse=True) #reverse옵션 넣어 역순 가능
staticmethod()
str()　　#내용물을 문자열로 반환
sum(자료형)　#리스트,튜플의 모든 요소의 합을 돌려줌
super()

T
tuple()　#튜플로 반환
type()　#변수의 유형 알려줌

V
vars()

Z
zip()　#자료형을 묶어줌

<2장 자료형>
2-1숫자형
 -정수형(Integer)
 -실수형(Floating-point)
 -x ** y　　x^y
 -x % y　　x/y 의 나머지 (tip 배수, 짝홀수 찾을 때)
 -x // y　x/y 의 몫

2-2문자형 =“”
 -문자열(String)
 -문자열 입력 ‘작은따옴표’ 말고 “큰따옴표” 쓰기
 -“““긴문자열””” (엔터 줄 바꿈도 기억)
 -\n　#new line 줄 바꿈
 -키보드에 백슬래시(\) 없으면 ₩로 대체
 -인덱싱[숫자] #그 순서에 해당하는 알파벳 (시작: 0)
 -슬라이싱[숫자:숫자] #그 순서에 해당하는 문자열 (끝 숫자 포함 X)
 -문자열 포매팅
  문자열 일때 “ %s ” %s
  정수 일때 “ %d ” %d 또는 %(d-300) 등
  ☆f-string　예시 day=10일 때
   f“today is {day}.” 결과는 today is 10.
 -int( )　　#문자열을 정수로
 -float( )　#문자열을 실수로
 -문자열 요소 대체 a.replace(“수정전”,“수정수”)

-문자열 관련 함수 (문자열에 쓰는 함수 .함수())
-문자 개수 세기 .count
-위치 알려주기1 find
-위치 알려주기2 index
-문자열 삽입 join
-소문자를 대문자로 바꾸기 .upper()
-대문자를 소문자로 바꾸기.lower()
-왼쪽 공백 지우기 .lstrip()
-오른쪽 공백 지우기 .rstrip()
-양쪽 공백 지우기 .strip()
-문자열 바꾸기 .replace(전, 후)
-문자열 나누기 .split(기준) 리스트로 반환
-문자열 첫 알파벳 대문자로 .capitalize()

2-3리스트 자료형
 -변수 = [,]
 -[:] #리스트 전체를 가리킴
 -a[x]　#리스트a의 x번째 요솟값
 -a[x]=y #리스트a의 x번째 요소값을 y로 수정
 -del a[x] #리스트a의 x번째 요솟값을 삭제

-인덱싱 a[x]　#리스트a의 x번째 요소값
-슬라이싱 a[x:y] #x번째 요소부터 y-1번째 요소까지
-더하기 a=[1,2], b=[3,4]　#a+b는 [1,2,3,4]
-곱하기(반복하기)　#a*3는 [1,2,1,2,1,2]
-길이 구하기 #len(a)는 2 (리스트 요소 두개라는 뜻)

-리스트 관련 함수
 -리스트 마지막에 요소 추가 a.append(모든자료형)
 -리스트 특정위치에 b 삽입 a.insert(인덱스,요소)
 -리스트 순서대로 정렬 a.sort()
 -리스트 현재의 역순으로 뒤집기 a.reverse()
 -위치 알려줌 a.index(요소)
 -리스트에서 첫번째 x요소 제거 a.remove(x)
 -리스트 마지막 요소 끄집어내서 삭제 a.pop()
 -리스트 y번째 요소 끄집어내서 삭제 a.pop(y)
 -리스트에 포함된 요소 x의 개수 세기 **a.count(x)**
 -리스트 확장 a.extend([,])

2-4튜플 자료형
 -변수 = (,) 또는 괄호 없이 ,
 -튜플 요소 수정, 삭제 불가
 -a[x]　#튜플a의 x번째 요솟값

리스트와 동일한 방법이다.

-인덱싱 a[x] #리스트a의 x번째 요소값
-슬라이싱 a[x:y] #x번째 요소부터 y-1번째 요소까지
-더하기 a=[1,2], b=[3,4] #a+b는 [1,2,3,4]
-곱하기(반복하기) #a*3는 [1,2,1,2,1,2]
-길이 구하기 #len(a)는 2 (튜플 요소 두개라는 뜻)

 -튜플 언팩킹
 a, b, c = (0, 1, 'cake eater')
 print(a,b,c)는 0 1 2
 -개수가 다를때 튜플 언팩킹
 a, b, *c = (0, 1, 2, 3)
 print(a,b,c)는 0 1 [2, 3]

2-5 딕셔너리 자료형
-변수={Key1:Value1, Key2:Value2, ...}
 #key 변하지 않는 값, value 변or불변 값
 #a={1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
-딕셔너리는 요소 순서로 인덱싱 불가, a[x]사용 못함
-단, key와 value를 간접적으로 얻을 수 있는 방법
#f딕셔너리를 for문에 돌리면, key값을 기준으로 반복
 a={'x': 1, 'y': 2}
 for i in a:
    print(i , a[i])
#결과값은 x 1   key=x, value2
          y 2
-dic.keys()   dict_keys([, ,])
-dic.values()

2-6 집합 자료형 =set([, ,]) or set(“ ”)
 -중복을 허용하지 않는다!
 -순서가 없다!
 -교집합 s1 & s2 또는 s1.intersection(s2)
 -합집합 s1 | s2 또는 s1.union(s2)
 -차집합 s1 - s2 또는 s1.difference(s2)
 -집합s1에 원소 추가 s1.add(원소)
 -집합s1에 여러 원소 추가 s1.update([, ,])
 -집합s1에서 특정원소 제거 s1.remove()

2-7 불 (참/거짓) 자료형
 - a == b #내용물이 동일
 - a != b #내용물이 동일하지 않음
 - a is b #동일한 변수
-자료형의 참/거짓
  문자열의 길이가 0이면 false, 0보다 크면 true
  리스트의 길이가 0이면 false, 0보다 크면 true
 -while 조건문:
        수행할 문장
              조건문이 참인 동안 반복 수행
 -if 자료형:
        print( ) #위 자료형이 true 일 때 실행됨
  else:
        print( ) #위 자료형이 false 일 때 실행됨
 -bool(자료형) #비었으면 false/비지 않았으면 true

값   참 or 거짓
"python"참
""   거짓
[1, 2, 3]참
[]   거짓
()   거짓
{}   거짓
1   참
0   거짓
None   거짓

2-8 변수
 - b = copy(a) 또는 b = a[:] #a의 내용만 복사
 - b = a   #a와 b는 같은 변수(같은 주소)
 - a, b = x, y #변수 한번에 여러개 만들기
 - a, b = b, a #변수의 내용물 바꾸기

<3장 제어문>
3-1 if 조건문
 -if 조건문:
        실행문
    else:
        실행문
 -x in 자료형, x not in 자료형
 -pass #아무 일도 하지 않고 지나가는 것 실행
 -elif #(else if 줄임말) 다중조건 만듦, 이전 조건문이 거짓일 때 수행됨
 -if not   #True/False가 바뀜
 -문자열 포매팅과 혼합
  score=100
  if score>80: message=success
        print(f“You achieved {message}”)
#결과: You achieved success

3-2 while 반복문
 -while 조건문:
        수행할 문장
              #조건문이 참인 동안 반복 수행
 -treeHit = treeHit+1 또는 treeHit+=1
        #treeHit 에 1씩 더하라는 의미
 -int()   #괄호 안을 받음
 -input() #키보드 입력
 -int(input()) #키보드 입력을 받음

-<예시> int(input("숫자를 입력해주세요: "))
-break #while문 강제 종료 (안쓰면 무한루프)
-while True: #일부러 무한 루프를 만듦
-: continue #while문 맨 처음(조건문)으로 돌아감

3-3 for 문
-for 변수 in 리스트(또는 튜플, 문자열):
    수행할 문장1
    수행할 문장2
-: continue #for문의 맨 처음(조건문)으로 돌아감
-end=" "   #다음 줄로 넘기지 않고 빈칸 출력
-print(" ") #빈칸 출력 및 다음 줄로
-print();print() #세미콜론(;) 한줄에 여러 개 명령 작성

-★리스트 포함 (List Comprehension)
a = [표현식(항목포함) for항목 in반복객체 if조건문]
-a = [90, 25, 67, 45, 80] 일 때
 #a[0]의 결과값은 90, a[1]의 결과값은 25
-#print(숫자 ,"문자", 변수) 쉼표로 구분하여 여러 자료형 출력 가능

<4장 입력과 출력>
4-1 함수
-def #함수를 만들 때 사용하는 예약어
-파이썬 함수의 구조
def 함수명(매개변수):
    <수행할 문장1>
-return #함수의 결과 값을 돌려주는 명령어
-함수 호출 시 입력했던 함수 정의를 잘 맞춰서(변수등)

-함수 호출할 때
#입력값과 결괏값이 있는 함수
def 함수이름(매개변수1,매개변수2,...):
  <수행할 문장> 생략하고 return에 직접입력 가능
   return 결과값

결괏값 변수 = 함수이름(입력인수1,입력인수2, ...)

-#입력값이 없고 결괏값만 있는 함수
def 함수이름():
    <수행할 문장> 생략하고 return에 직접입력 가능
     return 결과값

결괏값을 받을 변수 = 함수이름()

-#결과값이 없는 함수
 #결괏값은 오직 return 명령어로 돌려받음
def 함수이름(매개변수1,매개변수2,...):
  <수행할 문장>

함수이름(입력인수1, 입력인수2, ...)

-#입력값도 결괏값도 없는 함수
def 함수이름():
  <수행할 문장>

함수이름()

-def 함수이름(*args): #매개변수,입력값갯수 임의로
#args는 임의의 개수의 숫자를 가지는 튜플이 됨
#따라서 함수는 함수이름(, , , ...)
-for i in args:
    result = result + i 또는 result+ = i
  #result에 i씩 더하라는 의미
-#매개변수 이름을 지정해서 입력해줄 때
def print_kwargs(**kwargs):
#kwargs는 임의의 개수를 갖는 딕셔너리가 된다.
#함수 print_kwargs(key=value , ...)
#print(kwargs)의 결과는 {key : value , ...}
-return하는 값이 두개 이상일 때 튜플로 return됨
-함수에서 return문을 만나면 바로 빠져나옴
-def 함수이름(a , b , c=true) 일 때, 매개변수 c에 인수(입력값)를 넣지 않으면 기본값인 true가 사용됨
-기본값을 지정하는 매개변수는 가장 마지막에 놓기
-함수 밖의 변수a ≠ 함수 안의 매개변수a
-global a #함수 밖에서 정의된 변수a 사용(전역변수)

4-2입력과 출력
-a=input("문자열") #사용자 입력을 받을 때 사용
    #입력에 숫자를 넣어도 문자열로 인식
-print() 끝나면 자동으로 줄 바꿈
-print("a""b""c") abc #콤마가 없으면 붙여서 출력
-print("a","b","c") a b c #콤마를 넣어야 빈칸 출력
-print("a", end=" ") #a출력후 줄바꿈 말고 빈칸출력

4-3 파일 읽고 쓰기
-f=open("파일명.확장자","읽기방법 3개중하나")
 r (ead) / w (rite덮어쓰기) / a (ppend내용추가)
 # f 라는 변수에 파일 객체가 할당됨. 변수 f를 통해서 해당 파일에 접근이 가능하게 됨.
-f.write(변수or자료형) #파일에 괄호안 내용 적음
-f.close()   #파일 닫기 (마지막에 쓰는 습관)
-f.readline() #한번쓰면 f가 가리키는 파일에서 첫번째줄 읽어들임, 두번 쓰면 두번째줄 읽어들임...
-모든 줄 읽어 들이고 싶을 때
for line in f:
    print(line)
또는

```
  while True:    #false 까지 무한루프
      l = f.readline()
      if not l:
          break
      else:        #안써도 됨
          print(line)
```
#무한루프로 한줄 씩 읽어 나가다가 아무것도 없는데서 readline을 시도하면 아무것도 없는 것(None)은 False로 인식 후 not None이 True기 때문에 break 실행.

-read 모드에서 파일을 읽는 법, 함수 3가지

f.readline() #한줄

f.readlines() #각 줄을 요소로 갖는 리스트

f.read() #모든 줄을 한 문자열로 (잘 사용X)

-.strip() #\n 은 제거하고 불러오고 싶을 때

```
lines=f.readlines()
for i in lines:
    i = i.strip()    #strip함수는 줄 끝의 \n 제거
```
-with문 #파일을 열고 닫는 것을 자동으로 처리

```
  with open(파일명.확장자,"모드") as fp:
  #with문 밑에서는 f 파일 안에서 작업
```

<5장 파이썬 날개달기>

5-1 클래스

-클래스 #반복되는 변수&매서드(함수)를 미리 정해놓은 틀(설계도). 변수와 함수 덩어리.

-pass #아무것도수행X문법,임시로 코드작성할때사용

-class 안에 있는 함수를 메서드라고 부름

-dir(instance명) #instatnce안의 변수&매서드 알려줌

-self #class에서 메서드를 선언할 때 항상 맨 처음 매개변수로 self. instance = self

```
class Human:
  def info(self, tall, weight):
        실행할 문장
        return
a=Human()
```
-instance(a)는 클래스의 메서드와 변수를 갖게 됨.

a의 메서드는 #메서드의 매개변수에 실제 인수 대입

```
    a.info(163,55)  #보통 클래스 밖에서 인수 넣을 때
```
a의 변수는

```
    a.tall        #보통 클래스 안 실행문에서 사용
    a.weight    #클래스의 instance들 마다 변수이름을 공통으로 쓴다는 장점
```

```
-class Fourcal:
   def setdata(self, x, y):       #self=a
     self.first=x                #a의 first변수 = x
     self.second=second
   def add(self):  #()만 써도 오류X. but 매서드 안에서 자기자신을 무조건 받아주는것 권장
     self.result=self.first + self.second
     return self.result
a=Fourcal()            #a는 Fourcal()로 찍어낸 것.
```
#a라는 변수에 '설계도(과자틀)Fourcal'로 찍어낸 객체(과자)를 넣는다. 즉, a라는 변수는 설계도(과자틀)로 만든 과자. 따라서 Fourcal에 들어있는 함수(메서드)가 다 들어있다.

```
a.setdata(1, 2)         #a의 setdata에 a, 1, 2 입력
```
#a는 Fourcal로 찍어낸 것이므로 Fourcal에 있는 함수(메서드)인 setdata를 써서

a는 self에 (인수로 지정할 필요 없음),

1은 매개변수 x에, 2는 매개변수 y에 들어감.

```
print(a.first)          #a의 first변수를 출력
print(a.add())          #a의 add함수 값을 출력
```
#a는 Fourcal로 찍어낸 것이므로 Fourcal에 있는 함수(메서드)인 add도 사용가능하다. 따라서 Fourcal의 함수(메서드)를 순서대로 거쳐서 a.add() 값 출력.

```
-class Fourcal:
  def __init__(self, x , y):
      self.first = x
      self.second = y
```
#__init__함수(생성자)는 처음시작하다는 의미로, 클래스로 찍어낸 a가 실행할 때 일단 쳐음으로 수행되는 함수.

```
a=Fourcal(1,2)       #__int__함수의 매개 변수에 a,1,2
```
#생성자 __int__가 바로 실행되려면 first와 second가 먼저 지정되어야 한다.

-class 자식클래스명(부모클래스명): #클래스 상속

#부모에 있는것은 자식이 상속받으면 다 똑같이 쓸 수O. 즉, 기존에 짠 class를 그대로 활용.

-메서드 오버라이딩 #또 부모의 메서드를 자식이 다시 정의하여 덮어쓰기 할 수O.

```
-class Fourcal:
        x = 1          #클래스 변수
        y = 2
    def __init__(self, x, y):       #객체 변수
        self.first = x
        self.second = y
```
-공통으로 쓸 때 클래스변수, 각각 객체마다 다르게 줘야할 때 객체변수에 인수 입력한다.

-클래스 변수 #클래스에 미리 선언해 놓은 변수, 클래스에 공통적으로 적용되는 변수

-Fourcal.first = 3 (클래스명.클래스변수 = 자료형)

#해당 클래스의 클래스변수에 자료형 입력

#클래스자체를 호출해서 클래스 변수를 변경할 수 있음

-클래스 상속

super( ).__init__(,)
자식클래스에서 부모클래스 내용을 사용하고 싶은 경우
super().부모클래스내용
  <예시>
```
class Animal( ):
    def __init__( self, name ):
        self.name = name

class Human( Animal ):
    def __init__( self, name, hand ):
        super().__init__( name ) #  부모클래스의 __init__ 메소드 호출
        self.hand = hand

person = Human( "사람", "오른손" )
```

5-2 모듈
 -모듈 #미리 만들어 놓은 .py파일 (함수,변수,클래스)
 -방법1
  import math
  math.sqrt(인수, ..)
  math.sin(인수, ..)
 -방법2
  from math import sqrt, log, sin, ... 또는 *
  sqrt(인수, ..) # *은보통 전체를 의미
  sin(인수, ..)

 -내장된 모듈 math에 들어있는 함수
  sqrt() #루트값 출력
  log() #자연로그
  log10() #상용로그
  sin() , cos() 등
 -dir(모듈명) #모듈이 갖고 있는 함수 보여줌
 -if __name__ == "__main__":
#모듈을 불려오면 모듈 안 내용이 처음부터 끝까지 한번 실행하게 되어 있음. 그때 모듈안에 test용 print문 등이 있다면 그 결과 값이 출력됨. 따라서 조건문을 적고 print문을 적으면 모듈을 불러왔을 때 출려되지 않음.

5-3 패키지
 -패키지 #모듈 여러개 모아놓은 것(=라이브러리)

5-4 예외 처리
 -예외처리 #오류가 발생했을 때 어떻게 할지 정하는 것
 -기본구조
```
try:    #오류가 발생할 수 있는 구문
except Exception as e:  #오류가 발생시 할 것 지정
else:    #오류가 발생하지 않을 때 할 것 지정
finally: #무조건 마지막에 실행할 것 지정
```

```
try:
    실행 코드
except:
    예외가 발생했을 때 수행할 코드
else:
    예외가 발생하지 않았을 때 수행할 코드
finally:
    예외 발생 여부와 상관없이 항상 수행할 코드
```

 -Exception   #모든 오류의 최상위 부모
 -raise 오류명 #해당 오류명의 오류를 발생시킴

5-5 내장함수, 명령어
 -내장함수 #파이썬에서 기본적으로 포함하고 있는 함수 (특정한 자료형들에 공용)

5-6 라이브러리

<6장 프로그래밍>
 -프로그래밍은 입력과 출력 관점에서 생각하기
  입력 받는 값은?
  출력 받는 값은?
  생각해 볼 것은?

  함수 이름은?
  결과는 어떤 형태로?