3주차 수업 내용

- 자료형 간단 소개
- 제어문
- 조건문
- 반복문
- 기타 제어문

리스트(list)

- 여러 개의 데이터를 저장할 필요가 있을 때 리스트(list)를 사용
- 대괄호 사용해 묶음
- 리스트명 = [요소1, 요소2, 요소3, ...]
- L = [l1, l2, l3]

튜플(tuple)

- 여러 개의 객체를 모아 담는 데 튜플(tuple)을 사용
- 튜플은 리스트와 비슷하지만, 리스트 클래스에 있는 여러 기능이 없다.
- 소괄호 사용해 묶음. 생략도 가능
- tuple = (요소1, 요소2, 요소3, ...)
- T = (t1, t2, t3)
- 튜플은 수정이 불가능하며(immutable), 그래서 주로 문자열과 같이 바꿀 수 없는 객체들을 담을 때 사용됨
- 요소가 하나인 튜플을 만들 때는 요소 뒤에 쉼표(,)를 붙여 만듬

```
>>> t3 = 1, #요소가 한 개인 경우 ,를 붙인다
t3 = (1,)
```

dictionary(딕셔너리, 사전)

- 사전이나 전화번호부와 비슷한 자료구조
- 쉼표 기호(,)로 구분되는 키(key):값(value)의 쌍이 순서 없이 중괄호{ }로 묶인 자료구조
- 여러 개의 데이터 쌍(키:값)을 저장할 필요가 있을 때 딕셔너리를 사용
- dict = {키1:값1, 키2:값2, 키3:값3, ...}
- height = {'Park':174, 'Kim':170, 'Lee':165}

set(세트, 셋, 집합)

- 집합(set, 세트) 자료형은 중괄호 안에 요소들이 중복되지 않고, 순서없이 모여 있는 것
- 키만 모아 놓은 딕셔너리의 특수한 형태
- 세트를 생성하려면 딕셔너리처럼 중괄호를 사용하지만 콜론(:) 없이 값을 입력
- 집합은 포함된 요소들의 순서나 중복에 상관없이 묶음 자체를 필요로 할 때 주로 사용한다.
- 집합의 연산 등에 사용되며, 요소의 중복이 허용되지 않는 특징을
  이용해 리스트 또는 튜플의 중복 제거에 사용
- Set = {v1, v2, v3, ...}
- S = {s1, s2, s3}

학습 목표

- 조건문(`if` , `elif` , `else`)을 활용한 프로그램 흐름 제어
- 반복문(`for` , `while`)을 사용한 반복 작업 수행
- 실제 예제를 이용해 제어문의 활용 이해

프로그램 제어문

- 보통 프로그램은 위부터 순서대로 실행됨 - 순차문
- 계산의 순서를 바꾸는 여러 종류의 제어 문장이 있음
- 프로그램의 실행 순서를 바꾸는 대표적인 제어문
  - 조건문 : if, if-else, if-elif-else
  - 반복문 : while, for
  - 기타 제어문
    - break
    - continue
    - return

조건문

- 특정한 조건에 따라 프로그램의 실행 순서가 바뀌는 문장
- 조건문에는 if 문, if-else 문, if-elif-else 문이 있다

조건문: if 문

- 주어진 조건(condition)이 참(True)이면 블록(statement(s))을 실행하고, 거짓(False)이면 블록을 건너뛴다. (**블록** : 같은 들여쓰기 레벨을 갖는 문장들의 모임)

```
if condition :
    statement(s)
```

- 사용 예

```
score = int(input("정수 점수 입력:"))
if score >= 90 : #if문 끝에 :을 붙인다
    print("성적 : A") #들여쓰기
    print("장학금 수여")
print('welcome')
```

if 문 flowchart LR

%%{init: {'theme':'default'}}%% flowchart LR A1([start]) --> B[/score 값 받음/] B --> C{score >= 90} C -->|True| D[A, 장학금] D --> E['welcome'] C -->|False| E E --> F([end])

if 문 flowchart TD

%%{init: {'theme':'default', 'flowchart': {'curve': 'linear'}}}%% flowchart TD B[/score 값 받음/] B --> C{score >= 90} C -->|True| D[A, 장학금] D --> E['welcome'] C -->|False| E

조건문: if-else 문

- 주어진 조건(condition)이 True이면 block1을 실행하고, False이면 block2를 실행하는 문장

```
if condition :
    block1
else :
    block2
```

- 사용 예

```
num = int(input("Type int number:"))
if num > 0 :
    print("positive")
else :
    print("negative or zero")
```

중첩 if문(중복 if 문)

- 중첩 if 문은 if 문의 if 블록 안에 다른 if 문이 있거나 else 블록 안에 다른 if 문이 있는 문장
- 예: 임의의 정수를 입력하면, 그것이 양수인지, 음수인지, 0인지 판별해 출력하는 프로그램

- 작성 예 1

```
n = int(input("type int number : "))
if n >= 0 :
    if n == 0 :
        print("0")
    else :
        print("positive")
else :
    print("negative")
```

- 작성 예 2

```
n = int(input("type int number : "))
if n > 0 :
    print("positive")
else :
    if n < 0 :
        print("negative")
    else :
        print("0")
```

조건문: if-elif-else 문

- 3 가지 선택지 중 하나를 택할 때 사용

```
if condition1 :
    block1
elif condition2 :
    block2
else :
    block3
```

- condition1이 True이면 block1을 실행
- condition1이 False이고, condition2이 True 이면 block2를 실행
- 그 이외의 모든 경우는 block3를 실행
- 필요하다면 중간의 elif를 여러 개 추가 가능
- 필요한 경우에 마지막 else 생략 가능

사용 예제

- 앞의 작성 예 2를 다시 쓴 것

```
n = int(input("type int number : "))
if n > 0 :
    print("positive")
elif n < 0 :
    print("negative")
else :
    print("0")
```

앞의 중첩 if문 다시 쓴것

```
grade=int(input("점수를 입력하세요:"))
if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
elif grade >= 70:
    print("C")
elif grade >= 60:
    print("D")
else:
    print("F")
```

조건연산자 (또는 한 문장의 if-else문)

다음과 같이 두 수 중 더 큰 값을 결정하는 프로그램을

```
x=10
y=20
if x > y:
    max = x
else:
    max = y
print(max)
```

위처럼 여러 줄의 if-else 대신 다음과 같이 한 문장의 간결한 if-else 문을 사용할 수 있다.

```
x=10
y=20
max = x if x > y else y
print(max)
```

윤년 판정 프로그램

- 임의의 정수 y를 입력해 윤년인지 평년인지 판정하는 프로그램
- y가 다음 조건을 만족하면 윤년임
  - y가 4로 나누어 지면서 100으로는 나누어 지지 않거나
  - y가 400으로 나누어 지면

```
year = int(input("Type a year :"))
if (____________ and ____________) or ______________ :
    print(year, "is a leap year")
else :
    print(year, "is not a leap year")
```

- (해답)

반복문(Loops)

- 동일하거나 유사한 일을 여러 번 반복할 때 반복문을 사용하면 편리
- `*` 문자를 10번 출력? `print("*")` 문장을 10번 호출(X) 반복문을 사용하는 것이 편리(O)
- 파이썬 언어의 반복문은 `while`과 `for`의 두 종류
- while은 조건을 확인해 사용하는 반복에 적합

```
i = 0
while i < 10:
    print(i)
    i += 1
```

- for는 횟수가 정해진 반복문과 순서형 자료형의 이용에 적합

```
for i in range(10):
    print(i)
```

`while` 문

- 조건이 True인 동안 코드 블록을 반복 실행
- 기본 구조:

```
초기화
while 조건:
    블록
다음문장
```

- [초기화] 부분에서 초기값을 정한 후
- [조건]을 계산하여
  - 그 값이 True이면 반복문의 몸체인 [블록]을 수행
  - 값이 False이면 while 문을 벗어난 [다음문장]을 수행
- (문장의 수행을 무한 반복하지 않기 위해) 블록의 끝 부분에 조건의 변화가 있는 경우가 대부분

while 예제

- 0~9까지 출력

```
i = 0
while i < 10:
    print(i)
    i += 1
```

- count down 예제

```
count = 5
while count > 0:
    print(count)
    count -= 1
```

for 반복문

- 반복 가능한 객체(range, 리스트, 튜플 등)의 각 요소를 순회
- 기본 구조:

```
for 변수 in 반복_가능한_객체:
    실행할 코드
```

- 예시:

```
# print를 이용하여 *를 10번 출력
for i in range(10):
    print("*")
```

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

range() 객체의 사용

- range() 함수라고도 하지만 실제로는 클래스

```
>>> type(range(10))
<class 'range'>
```

- `for i in range(start, end, step):`
- `range()` 함수는 3개의 매개변수가 있는 데 `start`에서 시작해 `end`의 바로 전까지 `step` 간격으로 변수값을 생성
- `start`와 `step`의 기본값은 0과 1이기 때문에 생략 가능
- range(10) 또는 range(0, 10)이라고 호출하면 range(0, 10, 1)이라 호출한 것과 동일
- 즉 인수가 하나면 `end`가 주어진 것이고, 인수가 둘이면 `start`와 `end`가 주어진 것

예제

- 1부터 100까지의 합을 구하여 출력. (합 계산시 초기치 = 0)

```
sum = 0
for i in range(1, 101):
    sum = sum + i
print("sum is", sum)
```

- n!을 계산. (곱 계산시 초기치 = 1)

```
n = int(input('type int #:'))
prod = 1
for i in range(1, n+1):
    prod = prod * i
print("n! is ", prod)
```

무한 루프(endless loop)

- 반복문 수행 시 조건의 확인 부분이 항상 True이면 그 반복문은 무한히 실행됨
- while문으로 구현 가능

```
while True:
    print('1')
```

- 종료하려면 Ctrl+C를 눌러 빠져 나감

break

- loop를 중간에 벗어나고 싶을 때 사용
- 반복문의 수행 도중 어떤 조건에 다다르면 반복문에서 빠져 나오고 싶을 때 사용
- 중첩된 반복문에서는 break를 포함한 가장 가까운 반복문에서만 빠져나올 수 있음
- 사용 예: 구구단 출력 프로그램인데, 앞과 뒤의 숫자가 같으면 그 단의 출력을 멈추고 그 다음 단을 출력함

```
for n in range(2, 10):
    for x in range(1, 10):
        print(f"{n}*{x}={n*x}")
        if n == x:
            break
```

continue

- loop의 그 다음 차례 반복
- continue 문은 반복문의 수행 도중에 어떤 조건에 다다르면 반복문에서 빠져 나오는 대신 나머지 계산을 생략하고 그 다음 차례의 계산을 계속하게 함
- 다음은 n=10부터 n= -10까지의 수의 역수(1/n)를 구하는 프로그램이다. n이 0이면 역수 계산을 못하므로 continue 문을 사용하면 0의 역수만 계산하지 않고 그 다음 수의 역수를 계속 계산 가능

```
n = 10
while n >= -10 :
    if n == 0 :
        continue
    inv = 1.0 / n
    print(inv)
    n = n - 1
```

return

- 보통 함수의 맨 끝 부분에서 사용
- 호출한 함수로 돌아갈 때 반환값 없이 그냥 돌아갈 수도 있고 (이때는 꼭 return 문을 사용하지 않아도 됨)
- 또는 어떤 계산 값을 가지고 돌아갈 수도 있다.

```
return
return val
```

반복문과 함께 사용하는 else 문

- 드문 경우이기는 하지만 else 문을 반복문과 함께 사용할 수도 있다.
- 다음은 임의의 세 정수를 입력받는 for 문에서 음수가 한번도 입력되지 않는 경우에 break 문이 실행되지 않는다. 이 경우에 else 문이 실행됨 _

```
for i in range(3):
    n = int(input('Type integer:'))
    if n < 0:
        print('negative number entered')
        break
else: # break가 호출되지 않을 때만 실행됨
    print('No negative number entered.')
```

반복문 사용 가능 예제

- 임의의 수 범위에 있는 짝수, 약수 등의 출력
- 반복문 내에 조건문 사용
- 1부터 n까지의 합
- n의 약수 모두 출력, 약수의 갯수 출력, 약수의 합 등
- n^m의 계산, n!의 계산
- 주사위 n번 던지기
- 소수(prime number) 판정
- 1원부터 시작해 금액을 하루에 2배씩 늘려면서 누적 금액이 1억을 넘어서려면 며칠 걸릴까?