Week 2. Python 기초

들여쓰기 (Identation)

- 코딩블럭(기능을 가지는 코드 다발)을 표시하기 위해 들여쓰기(Identation)를 사용
- python의 특징으로 가독성이 높음, 하지만 세심한 주의를 요함
- 콜론 `:` 을 사용하고 내부의 코딩블럭은 동일한 들여쓰기(4개의 공백)를 함

(흐름)제어문

조건문 if

`if ~ else` 명령을 사용하면 조건에 따라 다른 명령을 수행할 수 있다.

- 조건문을 테스트해서 참이면 if문 바로 다음의 문장(if 블록)들을 수행하고
- 조건문이 거짓이면 else문 다음의 문장(else 블록)들을 수행함
- 그러므로 else문은 if문 없이 독립적으로 사용할 수 없음.

```
if condition:
    condition이 참일 때 실행되는 명령
else:
    condition이 거짓일 때 실행되는 명령
```

이 때 주의할 점은 참 또는 거짓일 때 실행되는 명령들은 빈칸 4칸 띄우고(또는 `tab` 한 번) 써야함.

if 조건문에서 "조건문(condition)"이란?

참과 거짓을 판단하는 문장.자료형의 참과 거짓에 대해서 몇 가지만 다시 살펴보면 다음과 같은 것들이 있음

자료형	참	거짓
숫자	0이 아닌 숫자	0
문자열	"abc"	""
리스트	[1,2,3]	[]
터플	(1,2,3)	()
딕셔너리	{"a":"b"}	{}

In [1]:
```
a = 5

if a > 1:
    print(a, "는(은) 1 보다 크닷!")
else:
    pass

print('--------------------')

######## if - else ###########
a = 7
b = 10
if (a % 2 == 0) or (b % 2 == 0):
    print("둘중에 적어도 하나는 짝수")
else:
    print("하나는 홀수")

print('end')
```

```
5 는(은) 1 보다 크닷!
--------------------
둘중에 적어도 하나는 짝수
end
```

`if a > 1:` 다음 문장은 Tap 키 또는 Spacebar키 4개를 이용해 반드시 들여쓰기 한 후에 `print("a 는 1 보다 크닷!")` 이라고 작성해야 한다.

and, or, not

한번에 여러 조건을 동시에 판단하기 위해 사용하는 연산자로는 `and`, `or`, `not`이 있음

연산자	설명
x or y	x와 y 둘중에 하나만 참이면 참이다
x and y	x와 y 모두 참이어야 참이다
not x	x가 거짓이면 참이다

x in s, x not in s

더 나아가 파이썬은 다른 프로그래밍 언어에서 쉽게 볼 수 없는 재미있는 조건문들을 제공함

in	not in
x in 리스트	x not in 리스트
x in 튜플	x not in 튜플
x in 문자열	x not in 문자열

In [2]:
```
my_univ = 'unist'
univ_list = ['unist','kaist','postec','snu']

if my_univ in univ_list:
    print('명단에 있습니다.')
else:
    print('명단에 없습니다.')
```

```
명단에 있습니다.
```

In [3]:
```
# 위 구조로도 if문의 `조건문`으로 사용할 수 있음
print(1 in [1, 2, 3])
print('j' in 'python')
```

```
True
False
```

조건이 여러 개일 때에는 elif

문제

만약 10문제를 보는 시험에서 8개 이상을 맞으면 성적이 "A", 8개 미만이고 5개 이상을 맞으면 "B", " 5개 미만을 맞으면 "C"가 된다면 점수는 다음과 같이 계산할 수 있다.

In [4]:
```
student = 10

if student >= 8:
    print("A")
elif (5 <= student < 8) :
    print("B")
else:
    print("C")
```

```
A
```


pass 할 때도 있어야지

In [5]:
```
x = 0

if x < 0 :
    print('negative!')
elif x == 0 :
    # TODO : 나중에 뭔가 추가 할 수 있다
    pass          # pass는 파이썬의 아무것도 하지 않음을 나타냄
else :
    print('positive!')
```

if 조건문의 간단한 표현(삼단표현)

if-else 블록을 한 줄로 표현 가능

```
varable = "true-expr" if "condition" else "false-expr"
```

In [6]:
```
x = 5

a = 'Non-negative' if x >= 0 else 'Negative'
print(a)
```

```
Non-negative
```

범위! range

`range(x)` 는 0부터 x-1 숫자만큼 1씩 증가하는 `리스트` (값 뭉텅이)를 만드는 함수

In [22]:
```
print('range(10):', range(10))
print(type(range(10)))
print(list(range(10)))
print('--------------------')

print('range(0,10,2):', range(0,10,2))
print(list(range(0,10,2)))
print(list(range(10,0,-2)))
print('--------------------')

######################
for i in range(0,10,2):
    print(i)

print('--------------------------------')

############################
seq = [0, 2, 4, 6, 8]
for i in range(len(seq)) :
    val = seq[i]
    print(val)
```

```
range(10): range(0, 10)
<class 'range'>
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
--------------------
range(0,10,2): range(0, 10, 2)
[0, 2, 4, 6, 8]
[10, 8, 6, 4, 2]
--------------------
0
2
4
6
8
--------------------------------
0
2
4
6
8
```


반복문 for

파이썬에서는 명령이 반복될 수 있게 하는 `for` 반복문을 사용할 수 있다. 반복문은 영어로 loop라고 한다.

```
for '카운터변수' in 'collection' :
    '반복해서 실행할 명령'
```

In [8]:
```
for a in [1,2,3,4]:
    print(a)
```

```
1
2
3
4
```

아래 코드의 의미는

- `[1, 2, 3]`이라는 리스트의 앞에서부터 하나씩 꺼내어 `a`라는 변수에 대입한 후 `print(a)`를 수행하라"이다.
- 당연히 `a`에 차례로 `1, 2, 3`이라는 값이 대입되며 `print(a)`에 의해서 그 값이 차례대로 출력된다.

In [9]:
```
for i in range(10):
    print("=" + str(i+1) + "=")
print('end')
```

```
=1=
=2=
=3=
=4=
=5=
=6=
=7=
=8=
=9=
=10=
end
```

문제

for 반복문과 문자열 연산을 사용하여 다음과 같이 인쇄해보렴.

```
*
**
***
****
*****
******
*******
********
*********
**********
```

In [10]:
```
for i in range(10):
    print("*"*(i+1))
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

중첩 for 반복문

- 반복문 안에서 다시 반복문을 실행해야 하는 경우
- for 반복문 안에 다시 for 반복문을 사용하는 중첩 반복문(nested loop)

문제

1부터 6까지의 눈금이 있는 주사위를 두 번 던져서 나온 모든 경우의 수를 표시해보자

In [11]:
```
for i in range(6):
    for j in range(6):
        print(i+1,j+1)
```

```
1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
2 5
2 6
3 1
3 2
3 3
3 4
3 5
3 6
4 1
4 2
4 3
4 4
4 5
4 6
5 1
5 2
5 3
5 4
5 5
5 6
6 1
6 2
6 3
6 4
6 5
6 6
```


문제 - 좀 더 어렵게

만약 주사위를 두 번 던져서 나온 숫자들의 합이 4의 배수가 되는 경우만 구해야 한다면

In [12]:
```
for i in range(6):
    n1 = i+1
    for j in range(6):
        n2 = j+1
        n_sum = n1 + n2
        if n_sum % 4 == 0:
            print(n1, n2)
        else:
            pass
```

```
1 3
2 2
2 6
3 1
3 5
4 4
5 3
6 2
6 6
```

반복문 while

for문과 마찬가지로 반복해서 문장을 수행할 수 있도록 해준다.

```
while (조건문):
    (실행문)
```

In [13]:
```
i = 0

while i < 3:
    i += 1
    print(i)
```

```
1
2
3
```

- 위의 예제는 i 값이 3보다 작은 동안 i=i+1과 print(i)를 수행하라는 말이다.
- i=i+1이라는 문장은 i의 값을 1씩 더하게 한다. i 값이 3보다 커지게 되면 while문을 빠져나가게 된다

print(i, end=" ")와 같이 입력 인수 end를 넣어 준 이유

해당 결과값을 출력할 때 다음줄로 넘기지 않고 그 줄에 계속해서 출력 할 수 있음.

조건에 맞지 않는 경우 맨 처음으로

while문을 빠져나가지 않고 while문의 맨 처음(조건문)으로 다시 돌아가게 만들고 싶은 경우가 생김.

이때 사용하는 것이 바로 `continue` 임

In [14]:
```
a = 0

while a < 10:
    a = a+1
    if a % 2 == 0:
        continue
    print(a, end='\n')
```

```
1
3
5
7
9
```

- 1부터 10까지의 숫자 중 홀수만을 출력하는 예이다. a가 10보다 작은 동안 a는 1만큼씩 계속 증가한다.
- `if a % 2 == 0` (a를 2로 나누었을 때 나머지가 0인 경우)이 참이 되는 경우는 a가 짝수일 때이다.
  즉, a가 짝수이면 continue 문장을 수행

while문 강제로 빠져나가기

while문은 조건문이 참인 동안 계속해서 while문 안의 내용을 반복적으로 수행함.

하지만 강제로 while문을 빠져나가고 싶을 때가 있는데 이 때 특정조건이 만족될 때 `break` 문을 쓰면 됨.

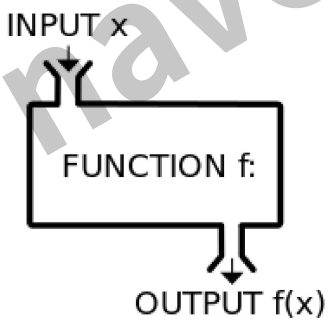
In [15]:
```
coffee = 3
money = 300

while money > 0:
    print("돈을 받았으니 커피를 줍니다.")
    coffee = coffee -1
    print("남은 커피의 양은 %d개입니다." %coffee)
    if coffee == 0:
        print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
        break
```

```
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 2개입니다.
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 1개입니다.
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 0개입니다.
커피가 다 떨어졌습니다. 판매를 중지합니다.
```

- `money`가 `false`가 아니니까 무한 루프를 돌게 된다.
- 그리고 while문의 내용을 한 번 수행할 때마다 `coffee = coffee - 1`에 의해서 coffee의 개수가 1개씩 줄어든다.
- 만약 `coffee`가 0이 되면 `if not coffee:`라는 문장에서 `not coffee`가 `True`이 되므로 if문 다음의 문장인 "커피가 다 떨어졌습니다. 판매를 중지합니다."가 수행되고 `break`문이 호출되어 `while`문을 빠져나가게 된다.

함수(Function)

함수(function)는 입력(input)을 받으면 그 입력에 해당하는 출력(output)을 내놓는 바로 그 무엇임




```
def 함수이름(입력변수):
    출력변수를 만드는 명령
    return 출력변수이름
```

- `def` (define)는 함수를 만들 때 사용하는 예약어이다.
- `return`은 정해진 함수의 역할을 다 하고 호출 한 곳으로 값을 돌려주는 역할을 한다.

In [16]:
```
def sum(a, b, c):
    d = a + b + c
    return d
```

In [17]:
```
first = 3
second = 4
third = 5

outcome = sum(first, second, third)
print(outcome)
```

```
12
```

- 위의 예제는 `sum` 이라는 함수를 만들고 그 함수를 어떻게 사용하는지를 보여준다. `sum(a, b)` 에서 a, b는 입력값이고, c (a+b)는 결과값이다.
- 즉 3, 4가 입력으로 들어오면 3+4를 수행하고 그 결과값인 7을 돌려(return) 준다.

문제

만약 10문제를 보는 시험에서 8개 이상을 맞으면 성적이 "A", 8개 미만이고 5개 이상을 맞으면 "B", " 5개 미만을 맞으면 "C"를 출력하는 함수 `grade` 를 만들어보세요.

짝수가 입력되면 "짝수"라는 문자열을, 홀수가 입력되면 "홀수"라는 문자열을 반환하는 함수 `evenOrOdd` 함수를 만들어보세요.

In [18]:
```
def grade(student_score):
    if student_score >= 8:
        dummy =  "A"
    elif (5 <= student_score < 8) :
        dummy = "B"
    else:
        dummy = "C"

    return dummy


final = grade(9)
```


지역(local variable)과 전국(global variable)

- 지역변수: 함수 안에서 만들어지고 사용되는 함수를 지역 변수, 영어로 local variable
  - 함수에 넣은 입력 변수나 함수 안에서 만들어진 변수는 함수 바깥에서는 사용할 수 없다.
  - 혹시 이름이 같은 변수가 있다고 하더라도 별개의 변수가 된다.

- 전역변수 : 지역 변수와 반대로 함수 바깥에서 만들어진 변수는 함수 안에서도 사용 가능, 영어로 global variable
  - 함수에 넣은 입력 변수나 함수 안에서 만들어진 변수는 함수 바깥에서는 사용할 수 없다.
  - 혹시 이름이 같은 변수가 있다고 하더라도 별개의 변수가 된다. 만약 바깥의 변수와 같은 이름의 변수를 함수 안에 다시 만들면 함수 안에서는 그 지역 변수를 사용하다가 함수 바깥으로 나오면 지역 변수는 사라지고 원래의 변수값으로 되돌아 온다.
  - 따라서, 함수 안에서는 함수 바깥에 있는 전역 변수의 값을 바꿀 수 없다.

In [23]:
```
z = 3

def globalization(x):
#     global z
    z = 99
    y = z * x
    print("함수 안의 변수 y =", y)
    print("함수 안의 변수 z =", z)
    return y

c = globalization(10)

# 함수 바깥에서는 여전히 z = 3
print('함수 밖의 변수 z =',z)
```

```
함수 안의 변수 y = 990
함수 안의 변수 z = 99
함수 밖의 변수 z = 3
```

만약 함수안에서 함수 바깥에 있는 변수의 값을 꼭 바꿔야만 한다면 다음과 같이 변수이름 앞에 `global` 키워드를 선언해 주면 된다.

예외처리

In [20]:
```
def attempt_float(x) :
    try:
        return float(x)
    except:
        return x

print(attempt_float('1.2'))          #float 로 변환 가능한 string
print(attempt_float('something'))    # float 로 변환 불가능한 string
```

```
1.2
something
```


나만의 함수 - 모듈

만약 내가 손수 만든 함수들이 있고, 그 것들이 다른 파일에 있다면 그 함수더미(`module`)을 불러와서 사용해아할텐데..

위의 외장함수를 사용하는 것 과 같이 `import` 를 이용하여 불러서 사용함

```
# mod1.py 라는 파일에 sum이라는 함수를 만들었다
def sum(a, b):
    c = a + b
    return c
```

위의 sum이라는 함수를 다른 파일들에서는 아래와 같이 사용함

```
import mod1
print(mod1.sum(3,4))

>>> 7
```

때로는 mod1. 조차 귀찮거나 저 파일을 다 부를 필요가 없을 때도 있음. 이때는

```
from mod1 import sum
sum(3, 4)

>>> 7
```

In [21]:
```
import math

a = math.sqrt(2)
print(round(a,3))
print(round(math.pi,2))
```

```
1.414
3.14
```