2주차 Python

- Relation Operators

Control statement -

if, else, elif

case(not in python)

- Loop statement -

for, while

MIT

6.0001 Introduction to Computer Science and Programming in Python

Fall 2016

CONTROL STATEMENT

- 일반적으로 컴퓨터 언어는 위에서부터 순차적으로 실행 됨. (이점 하드웨어와 다르다)
  - 당연히 예외가 필요함 (조건문, 반복)
- 조건: 원하는 조건을 만족할 때만 실행되도록 함
  - if, else, elif가 존재
- 반복: 조건을 두어서 반복하게 함
  - for – 미리 횟수를 알 수 있을 때 많이 사용
  - while – 매 loop 마다 더 해야 하는지 체크 (for 보다 범용)

참고로 Python에서는 C 언어의 case-switch (다중 분기) 문이 없다. if 문을 반복해서 사용할 수도 있고, 아니면 dictionary 구조를 이용하여 쉽게 가능하다.

BOOL (TRUE, FALSE 대소문자 구분)

논리값을 나타내는 자료형

False와 True값 중 하나를 가짐

a = False와 같이 이용

True와 False가 아닌 값을 Bool로 참, 거짓을 판단하는 상황에선 일반적으로 0 및 공백이 False, 0이 아닌 값을 True로 인식함

ex) a=1
if a:
실행됨

a=0
if a:
실행 안됨

```
a=True
if a:
    print("yes, true")

b=1
if b:
    print("yes, it is true, too")

#<- b 가 0 이 아니면 (음수라도) 모두 True로 취급
```

None 은 False로 취급이 된다.

COMPARISON OPERATORS

- Applied on int, float, **string**
- i and j are variable names
- comparisons below evaluate to a Boolean

```
i = "abc"
j = "def"
if i>j:
    print("abc is larger than def")
elif i==j:
    print('abc is equal to def')
else:
    print("def is larger than abc")
```

```
def is larger than abc
```

i > j

i >= j

i < j

i <= j

i == j → **equality** test, True if i is the same as j

i != j → **inequality** test, True if i not the same as j

LOGIC OPERATORS ON BOOLS

- `a` and `b` are variable names (with Boolean values)

`not a` → `True` if `a` is `False`
`False` if `a` is `True`

`a and b` → `True` if both are `True`

`a or b` → `True` if either or both are `True`

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

```
pset_time = 15
sleep_time = 8
print(sleep_time > pset_time)  -> False
drive = True
drink = False
print (drive and drink)  -> False
```

STRING COMPARISON

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

```
s1 = 'String'
s2 = 'String'
s3 = 'string'

if s1.casefold() == s3.casefold():
    print(s1.casefold())
    print(s3.casefold())
    print('s1 and s3 are equal in case-insensitive comparison')

if s1.lower() == s3.lower():
    print(s1.lower())
    print(s3.lower())
    print('s1 and s3 are equal in case-insensitive comparison')

if s1.upper() == s3.upper():
    print(s1.upper())
    print(s3.upper())
    print('s1 and s3 are equal in case-insensitive comparison')
```

```
string
string
s1 and s3 are equal in case-insensitive comparison
string
string
s1 and s3 are equal in case-insensitive comparison
STRING
STRING
s1 and s3 are equal in case-insensitive comparison
```

CONTROL FLOW – BRANCHING
(· INDENTATION에 주의)

```
if <condition>:
    <expression>
    <expression>
    ...
```

```
if <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

같은 indentation
C의 {
block {
이 해당

```
if <condition>:
    <expression>
    <expression>
    ...
elif <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

- <condition> has a value True or False
- evaluate expressions in that block if <condition> is True

INDENTATION

- matters in Python (tab or
- how you denote blocks of code

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
if x == y:
    print("x and y are equal")
    if y != 0:
        print("therefore, x / y is", x/y)
elif x < y:
    print("x  s smaller")
else:
    print("y  s smaller")
print("thanks!")
```

들여쓰기의 방법은 한칸, 두칸, 4칸, 탭 등 여러가지 방식이 있습니다.

중요한 것은 같은 블록 내에서는 들여쓰기 칸 수가 같아야 합니다. 위반시에는 "IndentationError: unexpected indent"라는 에러를 출력합니다.

IDE 나 Jupyter notebook 등에서는
: 이 나오면 다음 줄은 자동으로
indentation 이 됨.
(back indentation은 shift Tab)

PEP 8 -- STYLE GUIDE FOR PYTHON CODE

- https://www.python.org/dev/peps/pep-0008/#tabs-or-spaces
- Spaces are the preferred indentation method.
- Tabs should be used solely to remain consistent with code that is already indented with tabs.
- Python 3 disallows mixing the use of tabs and spaces for indentation.

RELATION OPERATORS 추가 (PYTHON편)

x in S : 집합(list, tuple, dictionary)및 string의 원소 가운데 x가 있으면 True

x not in S : 같은 조건에서 False

```
s = 'hello world'
if 'e' in s:
    print("h'e'lloworld")
```

NESTED IF

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

- 조건문 내부에 반복되는 조건문.

```
if score>60:
    if score >80:
        print("Top class")
    else:
        print("Good")
else:
    print("F")
```

좋은 PYTHON PROGRAM 습관

```
Don't
if a > 5:
        v = True
else:
        v = False

Instead
y = a>5    #it sets v to true if a > 5 else False

#True False 가 아닌 경우
v = 'Yes' if a > 5 else 'No'

v = ('No', 'Yes')[a>5]
```

- If 문 보다 logic operation 이 간단

```
if a == True:
        b = False
if a == False:
        b = True
# rather do this
b = not a
```

CASE(NOT IN PYTHON)

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

- 기존 switch-case문은 지정한 변수가 특정 값일때만 부분을 실행하도록 만들어진 if-else if문의 통합형 구조다.

(예시 : C언어)

```
switch(a){
    case 0:
        printf("a==0");
        break;
    case 1:  .....
    ...
    default:
        print("value error")
}
```

- python에서는 case문을 if~elif문의 반복으로 대체한다

```
if a==0:
   ~~~
elif a==1:
   ~~~
elif a==2:
...
```

== 와 IS (KEYWORD) 의 차이

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

==

value가 같은 가를 비교,

is

같은 변수인가를 따짐 (True, False)

```
x = ['apple', 'banana', 'cherry']
z = x
y = ['apple', 'banana', 'cherry']
print(x is y)      #False
print(x == y)    #True
print(z)
print(x is z)
-----
False
True
['apple', 'banana', 'cherry']
True
True
```

연습

변수 x, y, z를 테스트하여 셋 중 가장 큰 홀수 숫자를 저장하고 있는 변수를 출력하고, 세 변수 모두 짝수라면 이를 알리는 메시지를 출력한다.

LOOP

- for
- **know** the number of iterations

```
sum = 0
for i in range(11):
    sum +=i

print(sum)

55
```

- while
- **unbounded** number of iterations

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

CONTROL FLOW: WHILE LOOPS

```
while <condition>:
    <expression>
    <expression>
    ...
```

- <condition> evaluates to a Boolean
- if <condition> is True, do all the steps inside the while code block
- Expression에서 condition 관련 일을 수행하고, check <condition> again
- repeat until <condition> is False

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

55
```

WHILE LOOP(2)

- continue : while문의 현재 루프를 종료하고 다음 iteration 으로 감(while조건을 따지는 곳으로 돌아감)
- break : while문을 강제로 종료하고 while문 다음 명령어부터 계속함

```
while True:
    if flag_error==True:
        break
```

BREAK STATEMENT IN NESTED WHILE

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- exits only innermost loop!

```
while <condition_1>:
    while <condition_2>:
        <expression_a>
        break
        <expression_b>
    <expression_c>
```

작은 문제 (CONTINUE, BREAK 이해)

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

#아래를 돌리면 sum의 값이 얼마로 나올까요?

```
i=-1
sum = 0
while i <= 10:
    i+=1
    if i%2==0:
        continue
    else:
        sum+=i
        if sum>10:
            break


print ('while loop out', sum)
```

CONTROL FLOW: FOR LOOPS

```
for <variable> in range(<some_num>):
    <expression>
    <expression>
    ...
```

- each time through the loop, `<variable>` takes a value
- first time, `<variable>` starts at the smallest value
- next time, `<variable>` gets the prev value + 1
- etc.

RANGE(START,STOP,STEP)

- default values are `start = 0` and `step = 1` and optional
- loop until value is `stop - 1`

```
mysum = 0
for i in range(7, 10):
    mysum += i
print(mysum)
```

```
mysum = 0
for i in range(5, 11, 2):
    mysum += i
print(mysum)
```

예 (CONTINUE VS BREAK)

```
mysum= 0
for i in range(5, 9, 2):
    mysum+= i
    if mysum== 5:
        continue
    else:
        mysum+=1
print(mysum)
```

답: 13

```
mysum= 0
for i in range(5, 9, 2):
    mysum+= i
    if mysum== 5:
        break
    else:
        mysum+=1
print(mysum)
```

답: 5

FOR LOOP WITH ITERABLES

- C program 에서 for loop 의 index 는 숫자
- Python의 경우 숫자도 지원 (range(…)). 더 중요한 것은 iterable 을 이용한 loop 만들기 가능
- An iterable is anything you can loop over with a for loop in Python. Iterables can be looped over, and anything that can be looped over is an iterable. Sequences are a very common type of iterable. Lists, tuples, and strings are all sequences.

String

```
for s in "helloworld":
    print(s)
```

```
h
e
l
l
o
w
o
r
l
d
```

list

```
for i in [0,2,4,6,8]:
    print(i)
```

```
0
2
4
6
8
```

```
for i in [0, "hello", 9, "world"]:
    print(i)
```

```
0
hello
9
world
```

tuple

```
for i in (0,2,4):
    print(i)
```

```
0
2
4
```

```
sum=0
for i in range(10):
    sum += i
print(sum)
```

```
45
```

```
for i in range(10,100,11):
    print(i)
```

```
10
21
32
43
54
65
76
87
98
```

FOR LOOP WITH SET, DICTIONARY

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
--------------------
cherry
apple
banana
```

```
thisdict = {'a': 1, 'b': 2}

for x, y in thisdict.items():
  print(x, y)

for x in thisdict:
  print(thisdict[x])

for x in thisdict.values():
  print(x)

for x in thisdict.keys():
  print(x)
```

```
a 1
b 2
1
2
1
2
a
b
```

FOR VS while LOOPS

서울대학교 SEOUL NATIONAL UNIVERSITY

for loops

- **know** number of iterations
- can **end early** via `break`
- uses a **counter**
- **can rewrite** a `for` loop using a `while` loop

while loops

- **unbounded** number of iterations
- can **end early** via `break`
- can use a **counter but must initialize** before loop and increment it inside loop
- **may not be able to rewrite** a `while` loop using a `for` loop

실습

- 1~100번째까지의 피보나치 수열을 print해보기

The Fibonacci Sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13,….

The next number is found by adding up the two numbers before it:

the 2 is found by adding the two numbers before it (1+1),
the 3 is found by adding the two numbers before it (1+2),
the 5 is (2+3),
and so on!

과제

- 두 년도를 입력받아 두 해 사이에 몇일이 흘렀는지 계산하기
- 윤년을 고려해야 함 -> if문과 loop문을 적절히 이용

STRING – IN PYTHON

- think of as a **sequence** of case sensitive characters
- can compare strings with ==, >, < etc.
- `len()` is a function used to retrieve the **length** of the string in the parentheses
  - len('abc') -> 3

- 배울 중요한 것:
  - Indexing 방법 -> 뒤의 list, tuple 등에도 적용
  - Mutable (내용을 바꿀 수 있는 것) and immutable (바꿀 수 없는 것) 의 차이

STRINGS

- square brackets used to perform **indexing** into a string to get the value at a certain index/position

```
s = "abc"
```

index: 0 1 2 ← indexing always starts at 0

index: -3 -2 -1 ← last element always at index -1

```
s[0]    → evaluates to "a"
s[1]    → evaluates to "b"
s[2]    → evaluates to "c"
s[3]    → trying to index out of bounds, error
s[-1]   → evaluates to "c"
s[-2]   → evaluates to "b"
s[-3]   → evaluates to "a"
```

STRINGS (뒤에 나오는 LIST도 동일)

- can **slice** strings using `[start:stop:step]`
- if give two numbers, [`start:stop`], `step=1` by default
- you can also omit numbers and leave just colons

```
s = "abcdefgh"
```

`s[3:6]` → evaluates to `"def"`, same as `s[3:6:1]`

`s[3:6:2]` → evaluates to `"df"`

If unsure what some command does, try it out in your console!

`s[::]` → evaluates to `"abcdefgh"`, same as `s[0:len(s):1]`

`s[::-1]` → evaluates to `"hgfedcba"`, same as `s[-1:-(len(s)+1):-1]`

`s[4:1:-2]` → evaluates to `"ec"`

STRING - IMMUTABLE

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

- Immutable: cannot change the content once created

```
s1='hello'

print(s1[0])

s1[0]='y'

print(s1)
```

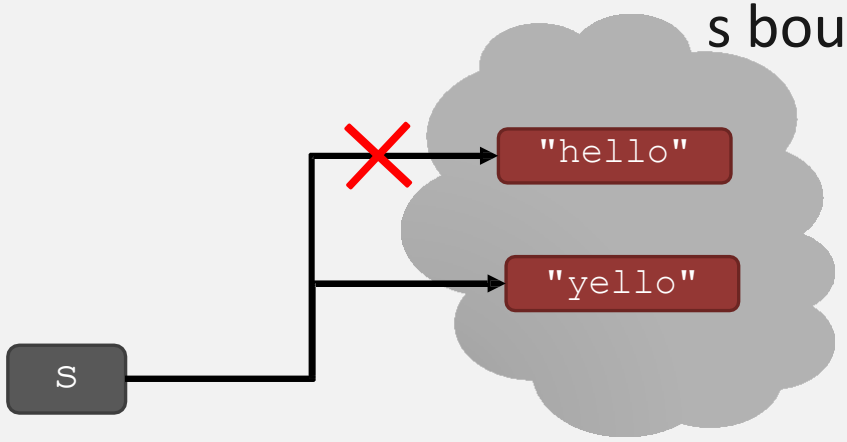
```
----------
h
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-8a501bfa2982> in <module>
      1 s1='hello'
      2 print(s1[0])
----> 3 s1[0]='y'
      4 print(s1)

TypeError: 'str' object does not support item assignment
```

STRINGS - IMMUTABLE

- strings are "**immutable**" – cannot be modified

```
s = "hello"
s[0] = 'y'
```
→ gives an error

```
s = 'y'+s[1:len(s)]
```
→ is allowed,
s bound to new object



STRINGS AND LOOPS (중요)

- these two code snippets do the same thing
- bottom one is more “pythonic”

```
s = "abcdefgh"
for index in range(len(s)):
    if s[index] == 'i' or s[index] == 'u':
        print("There is an i or u")
```

```
for  char in s:
            if char == 'i' or char == 'u':
                print("There is an i or u")
```

EXERCISE

```
s1 = "Seoul National Univ 한국"
s2 = "Samsung 한국"
for char1 in s1:
    for char2 in s2:
        if char1 == char2:
            print("common letter", char1)
            break
```

```
common letter S
common letter u
common letter
common letter a
common letter n
common letter a
common letter
common letter n
common letter
common letter 한
common letter 국
```

GUESS-AND-CHECK

- the process below also called **exhaustive enumeration**

- given a problem…
- you are able to **guess a value** for solution
- you are able to **check if the solution is correct**
- keep guessing until find solution or guessed all values

GUESS-AND-CHECK

서울대학교 SEOUL NATIONAL UNIVERSITY

– cube root

```
cube = 8
for guess in range(cube+1):
    if guess**3 == cube:
        print("Cube root of", cube, "is", guess)
```

GUESS-AND-CHECK

- cube root

```
cube = 8
for guess in range(abs(cube)+1):
    if guess**3 >= abs(cube):
        break
if guess**3 != abs(cube):
    print(cube, 'is not a perfect cube')
else:
    if cube < 0:
        guess = -guess
    print('Cube root of '+str(cube)+' is '+str(guess))
```

APPROXIMATE SOLUTIONS

- **good enough** solution
- start with a guess and increment by some **small value**
- keep guessing if `|guess**3-cube| >= epsilon`
  for some **small epsilon**

- decreasing increment size → slower program
- increasing epsilon → less accurate answer

APPROXIMATE SOLUTION

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

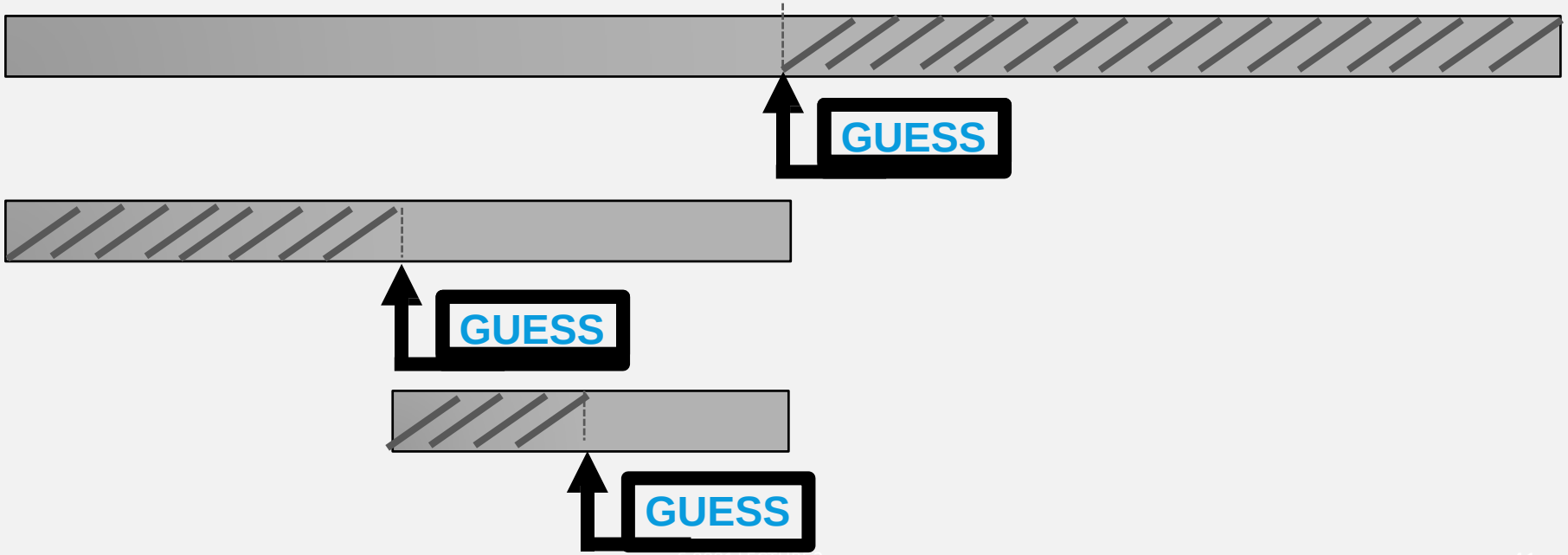
– cube root

```
cube = 27
epsilon = 0.01
guess = 0.0
increment = 0.0001
num_guesses = 0
while abs(guess**3 - cube) >= epsilon and guess <= cube :
    guess += increment
    num_guesses += 1
print('num_guesses =', num_guesses)
if abs(guess**3 - cube) >= epsilon:
    print('Failed on cube root of', cube)
else:
    print(guess, 'is close to the cube root of', cube)
```

num_guess, guess 29997 2.999700000001906

BISECTION SEARCH

- half interval each iteration
- new guess is halfway in between
- to illustrate, let's play a game!



BISECTION SEARCH – cube root

```
cube = 27
epsilon = 0.01
num_guesses = 0
low = 0
high = cube
guess = (high + low)/2.0
while abs(guess**3 - cube) >= epsilon:
    if guess**3 < cube :
        low = guess
    else:
        high = guess
    guess = (high + low)/2.0
    num_guesses += 1
print 'num_guesses =', num_guesses
print guess, 'is close to the cube root of', cube
```

um_guesses =, guess 14 3.000091552734375

BISECTION SEARCH CONVERGENCE

- search space
  - first guess: N/2
  - second guess: N/4
  - kth guess: $N/2^k$
- guess converges on the order of $\log_2 N$ steps
- bisection search works when value of function varies monotonically with input
- code as shown only works for positive cubes > 1 – why?
- challenges → modify to work with negative cubes!
  → modify to work with x < 1!

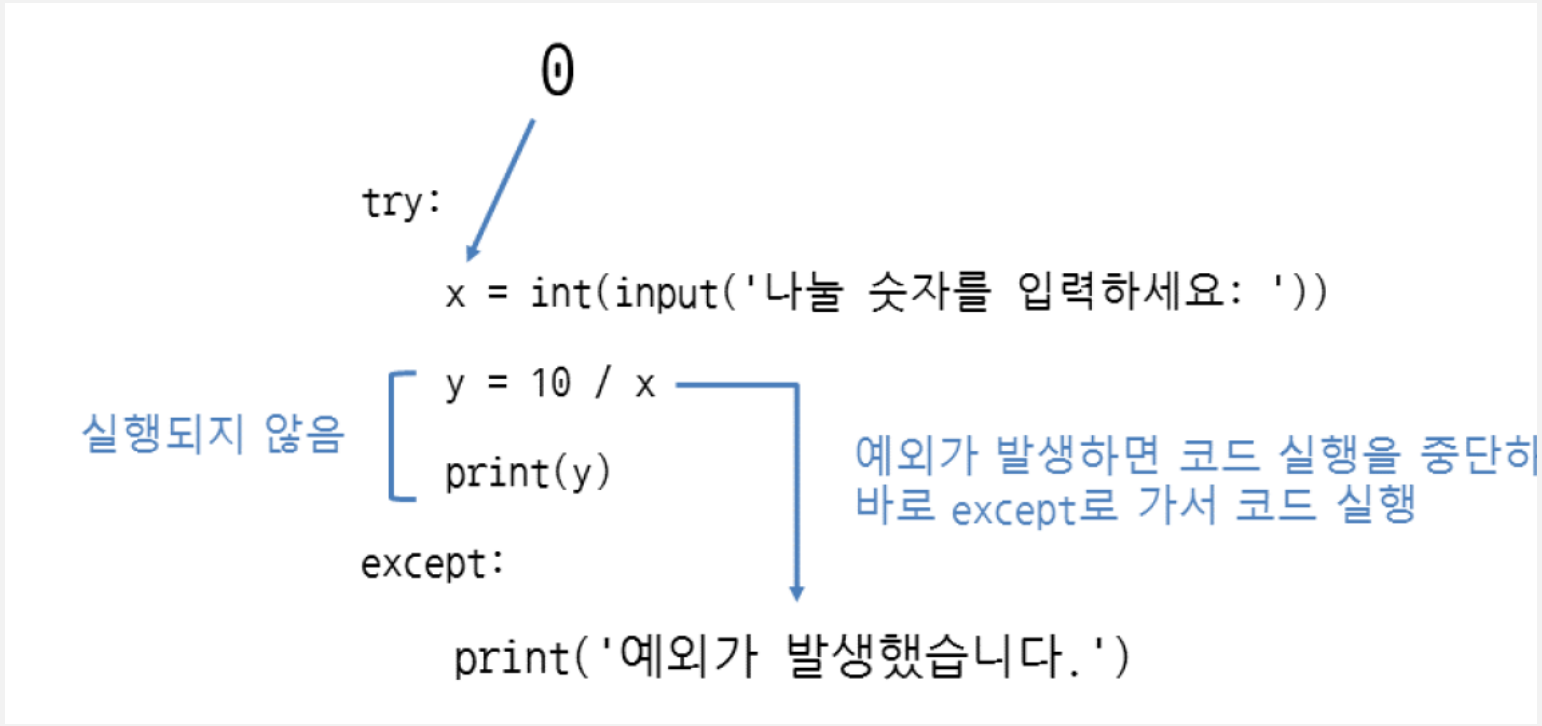
TRY
EXCEPT

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

try:
    실행할 코드
except:
    잘못되었을 때 실행코드

```
>>> ten_div(0)
Traceback (most recent call last):
  File "<pyshell#121>", line 1, in <module>
    ten_div(0)
  File "<pyshell#119>", line 2, in ten_div
    return 10 / x
ZeroDivisionError: division by zero
```

```
try:
    실행할 코드
except:
    예외가 발생했을 때 처리하는 코드
```



https://dojang.io/mod/page/view.php?id=2398

```
y = [10, 20, 30]

try:
    index, x = map(int, input('인덱스와 나눌 숫자를 입력하세요: ').split())
    print(y[index] / x)
except ZeroDivisionError:    # 숫자를 0으로 나눠서 에러가 발생했을 때 실행됨
    print('숫자를 0으로 나눌 수 없습니다.')
except IndexError:           # 범위를 벗어난 인덱스에 접근하여 에러가 발생했을 때 실
    print('잘못된 인덱스입니다.')
```

map(변환 함수, iterable 데이터)
map() 함수는 두번째 인자로 넘어온 데이터가 담고 있는 모든 데이터에 변환 함수를 적용하여 다른 형태의 데이터를 반환합니다.

```
# try-except 문
def safe_pop_print(list, index):
    try:
        print(list.pop(index))
    except IndexError:
        print('{} index의 값을 가져올 수 없습니다.'.format(index))

safe_pop_print([1,2,3], 5) # 5 index의 값을 가져올 수 없습니다.

# if 문
def safe_pop_print(list, index):
    if index < len(list):
        print(list.pop(index))
    else:
        print('{} index의 값을 가져올 수 없습니다.'.format(index))

safe_pop_print([1,2,3], 5) # 5 index의 값을 가져올 수 없습니다.
```

ERROR 이름 확인

```
# 에러 이름 확인
try:
    list = []
    print(list[0])  # 에러가 발생할 가능성이 있는 코드

except Exception as ex: # 에러 종류
    print('에러가 발생 했습니다', ex) # ex는 발생한 에러의 이름을 받아오
는 변수
    # 에러가 발생 했습니다 list index out of range
```

https://wayhome25.github.io/python/2017/02/26/py-12-exception/

```
# 올바른 값을 넣지 않으면 에러를 발생시키고 적당한 문구를 표시한다.
def rsp(mine, yours):
    allowed = ['가위','바위', '보']
    if mine not in allowed:
        raise ValueError
    if yours not in allowed:
        raise ValueError


try:
    rsp('가위', '바')
except ValueError:
    print('잘못된 값을 넣었습니다!')
```

문제

- f(x) = x**2 + 4*x + 3 의 최저값을 가지는 x의 위치, xmin과 그 때의 최저값을 구하라. 단, xmin은 -10과 +10사이에 있다.

- 문제 pi 값을 구하라.

X=0.00000~1.0000

Y = X=0.00000~1.0000

x**2 + y**2 <1.0000

이 개수의 비율에서 pi를 구한다.

```
str_x = input("type x")
str_y = input("type y")
str_z = input("type z")
x = int(str_x); y = int(str_y); z=int(str_z)
print(x, y, z)

if x%2==0 and y%2==0 and z%2==0:
    print("all variables are even numbers")

elif x%2==0 and y%2==0 and z%2!=0:
    print('z')

elif x%2==0 and y%2!=0 and z%2==0:
    print('y')

elif x%2!=0 and y%2==0 and z%2==0:
    print('x')

elif x%2==0 and y%2!=0 and z%2!=0:
    if y>=z:
        print('y')
    else:
        print('z')

elif x%2!=0 and y%2==0 and z%2!=0:
    if x>=z:
        print('x')
    else:
```