

Python Functions

- Basics of Python Functions
- Python Built-in Functions
- Problem Solving using Python Functions

일반적인 함수

입력값이 있고 결과값이 있는 함수가 일반적인 함수이다.
함수는 대부분 다음과 비슷한 형태일 것이다.

```
def 함수이름(입력인수):  
    <수행할 문장>  
    ...  
    return 결과값
```

```
def sum(a, b):  
    result = a + b  
    return result
```

```
>>> a = sum(3, 4)  
>>> print(a)  
7
```

입력값이 없는 함수

입력값이 없는 함수가 존재할까? 당연히 존재한다.

```
>>> def say():  
...     return 'Hi'  
...
```

```
>>> a = say()  
>>> print(a)  
Hi
```

결과값이 없는 함수

결과값이 없는 함수 역시 존재한다. 다음의 예를 보자.

```
>>> def sum(a, b):  
...     print("%d, %d의 합은 %d입니다." % (a, b, a+b))  
...
```

입력값도 결과값도 없는 함수

입력값도 결과값도 없는 함수 역시 존재한다.

```
>>> def say():  
...     print('Hi')  
...
```

여러 개의 입력값을 받는 함수 [1/2]

다음의 예를 통해 여러 개의 입력값을 모두 더하는 함수를 직접 만들어 보자. 예를 들어 `sum_many(1, 2)`이면 3을, `sum_many(1,2,3)`이면 6을, `sum_many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`이면 55를 돌려주는 함수를 만들어 보자.

```
>>> def sum_many(*args):  
...     sum = 0  
...     for i in args:  
...         sum = sum + i  
...     return sum  
...  
>>>
```

```
>>> result = sum_many(1,2,3)  
>>> print(result)  
6  
>>> result = sum_many(1,2,3,4,5,6,7,8,9,10)  
>>> print(result)  
55
```

여러 개의 입력값을 받는 함수 [2/2]

```
>>> def sum_mul(choice, *args):  
...     if choice == "sum":  
...         result = 0  
...         for i in args:  
...             result = result + i  
...     elif choice == "mul":  
...         result = 1  
...         for i in args:  
...             result = result * i  
...     return result  
...  
>>>
```

```
>>> result = sum_mul('sum', 1,2,3,4,5)  
>>> print(result)  
15  
>>> result = sum_mul('mul', 1,2,3,4,5)  
>>> print(result)  
120
```

함수의 결과값은 언제나 하나이다

먼저 다음의 함수를 만들어 보자.

```
>>> def sum_and_mul(a,b):  
...     return a+b, a*b
```

Tuple로 값을 return



```
>>> result = sum_and_mul(3,4)
```

```
>>> sum, mul = sum_and_mul(3, 4)
```

입력 인수에 초깃값 미리 설정하기

```
def say_myself(name, old, man=True):  
    print("나의 이름은 %s 입니다." % name)  
    print("나이는 %d살입니다." % old)  
    if man:  
        print("남자입니다.")  
    else:  
        print("여자입니다.")
```

. 초기화시키고 싶은 입력 변수들을 항상 뒤쪽에 위치시키는 것을 잊지 말자.

ARGUMENT TYPES

Regular
Argument

Keyword
Argument

def myFunc(var1, var 2 = 3):

...

Keyword args set
DEFAULT value that
MAY be overridden

```
def myFunc(var1, var2=3):  
    return var1 + var2
```

```
myFunc(10, 10)
```

```
myFunc(10)
```

함수 안에서 함수 밖의 변수를 변경하는 방법

```
# vartest_return.py
a = 1
def vartest(a):
    a = a + 1
    return a

a = vartest(a)
print(a)
```

Global 명령어를 이용하기

```
# vartest_global.py
a = 1
def vartest():
    global a
    a = a + 1

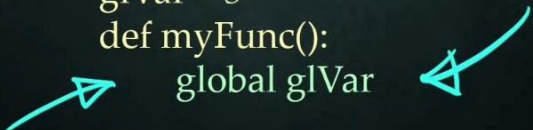
vartest()
print(a)
```

LOCAL VS GLOBAL VARIABLES

GLOBAL: variable that accessible
ANYWHERE within program.

Uses keyword 'global'

```
glVar = 5
def myFunc():
    global glVar
```



```
glVar = 5
```

```
def myFunc1():
    global glVar
    glVar = glVar - 10
    print("Current glVar: ", glVar)
```

```
def myFunc2():
    global glVar
    glVar = glVar + 10
    print("Current glVar: ", glVar)
```

```
myFunc1()
myFunc2()
```


COMMENTS

- Tell program to IGNORE everything afterward in line
- declared with '#' pound/sharp symbol
- Frequently used to write notes or 'ignore' bits of code

comment 1

x = 5 #2

#3

DOCUMENT STRING

- Text DESCRIBING the function
- Comes immediately after function creation
- Use triple quotes to enclose

```
def myFunc():
```

```
    """
```

```
    My description
```

```
    """
```

```
>>> print(myFunc.__doc__)
```

My description

```
>>> print(myFunc.__name__)
```

Python Special Variables

`__doc__`

`__name__`

Predefined Attributes

- Called “special variables” or “magic variables”
 - They contain meta-data about script files / modules
 - The form of `__<variable>__`, which is enclosed by two underscores
- One important variable is `__name__`
 - it tells us the **name** of the module
 - currently running script file will have `__name__ = "__main__"`

```
>> import math
>> math.__name__
'math'
>> __name__
'__main__'
```

- The complete list of predefined attributes are listed in <https://docs.python.org/2/reference/datamodel.html>
- `__name__`, `__dict__`, `__doc__`, `__code__`, 등등

suppose we have testFile.py as follows

```
def testFile(dest):  
    print(dest)  
  
if __name__ == '__main__':    # Is this the main file?  
    testFile('ham')  
    print('done!!')
```

=====

testFile.py를 Python interpreter에서 수행하면 (즉 `python testFile.py` 하면)
`if __name__ == '__main__':` 이 **true**가 되고 그 아래 문장들이 수행됨

반면에 `import testFile` 하면
`if __name__ == '__main__':` 이 **false**가 되고 그 아래 문장들이 수행이 안됨

****** `__name__` 은 python의 special variable로써 현재 수행되는 .py file의 상태정보가지고 있음
➔ outside module을 수행하는지, main module을 수행하는지

Python Functions

- Basics of Python Functions
- Python Built-in Functions
- Problem Solving using Python Functions

Python Built-in Functions [1/16]

		Built-in Functions		
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

abs

abs(x)는 어떤 숫자를 입력으로 받았을 때, 그 숫자의 절대값을 돌려주는 함수이다.

```
>>> abs(3)
3
>>> abs(-3)
3
>>> abs(-1.2)
1.2
```

all

all(x)은 반복 가능한(iterable) 자료형 x를 입력 인수로 받으며, 이 x가 모두 참이면 True, 거짓이 하나라도 있으면 False를 리턴한다.

```
>>> all([1, 2, 3])
True
```

리스트 자료형 [1, 2, 3]은 모든 요소가 참이므로 True를 리턴한다.

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

리스트 자료형 [1, 2, 3, 0] 중에서 요소 0은 거짓이므로 False를 리턴한다.

Python Built-in Functions [3/16]

any

`any(x)`는 `x` 중 하나라도 참이 있을 경우 `True`를 리턴하고, `x`가 모두 거짓일 경우에만 `False`를 리턴한다. `all(x)`의 반대 경우라고 할 수 있다.

다음의 예를 보자.

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

리스트 자료형 `[1, 2, 3, 0]` 중에서 `1, 2, 3`이 참이므로 `True`를 리턴한다.

```
>>> any([0, ""])
False
```

리스트 자료형 `[0, ""]`의 요소 `0`과 `""`은 모두 거짓이므로 `False`를 리턴한다.

chr

`chr(i)`는 아스키(ASCII) 코드값을 입력으로 받아 그 코드에 해당하는 문자를 출력하는 함수이다.

(※ 아스키 코드란 0에서 127 사이의 숫자들을 각각 하나의 문자 또는 기호에 대응시켜 놓은 것이다.)

```
>>> chr(97)
'a'
>>> chr(48)
'0'
```

ord

`ord(c)`는 문자의 아스키 코드값을 리턴하는 함수이다.

(※ `ord` 함수는 `chr` 함수와 반대이다.)

```
>>> ord('a')
97
>>> ord('0')
48
```

dir

`dir`은 객체가 자체적으로 가지고 있는 변수나 함수를 보여 준다. 아래 예는 리스트와 딕셔너리 객체의 관련 함수들(메서드)을 보여 주는 예이다. 우리가 02장에서 살펴보았던 자료형 관련 함수들을 만나볼 수 있을 것이다.

```
>>> dir([1, 2, 3])
['append', 'count', 'extend', 'index', 'insert', 'pop',...]
>>> dir({'1':'a'})
['clear', 'copy', 'get', 'has_key', 'items', 'keys',...]
```

divmod

`divmod(a, b)`는 2개의 숫자를 입력으로 받는다. 그리고 `a`를 `b`로 나눈 몫과 나머지를 튜플 형태로 리턴하는 함수이다.

```
>>> divmod(7, 3)
(2, 1)
>>> divmod(1.3, 0.2)
(6.0, 0.099999999999999978)
```


enumerate

enumerate는 "열거하다"라는 뜻이다. 이 함수는 순서가 있는 자료형(리스트, 튜플, 문자열)을 입력으로 받아 인덱스 값을 포함하는 enumerate 객체를 리턴한다.

(※ 보통 enumerate 함수는 아래 예제처럼 for문과 함께 자주 사용된다.)

무슨 말인지 잘 이해되지 않으면 다음 예를 보자.

```
>>> for i, name in enumerate(['body', 'foo', 'bar']):
...     print(i, name)
...
0 body
1 foo
2 bar
```

eval

eval(expression)은 실행 가능한 문자열(1+2, 'hi' + 'a' 같은 것)을 입력으로 받아 문자열을 실행한 결과값을 리턴하는 함수이다.

```
>>> eval('1+2')
3
>>> eval("'hi' + 'a'")
'hia'
>>> eval('divmod(4, 3)')
(1, 1)
```

보통 eval은 입력받은 문자열로 파이썬 함수나 클래스를 동적으로 실행하고 싶은 경우에 사용된다.

filter

filter란 무엇인가를 걸러낸다는 뜻으로, filter 함수도 동일한 의미를 가진다. filter 함수는 첫 번째 인수로 함수 이름을, 두 번째 인수로 그 함수에 차례로 들어갈 반복 가능한 자료형을 받는다. 그리고 두 번째 인수인 반복 가능한 자료형 요소들이 첫 번째 인수인 함수에 입력되었을 때 리턴값이 참인 것만 묶어서(걸러내서) 돌려준다.

다음의 예를 보자.

```
#positive.py
def positive(l):
    result = []
    for i in l:
        if i > 0:
            result.append(i)
    return result

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

결과값: [1, 2, 6]

filter 함수를 이용하면 위의 내용을 아래와 같이 간단하게 작성할 수 있다.

```
#filter1.py
def positive(x):
    return x > 0

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

결과값: [1, 2, 6]

Python Built-in Functions [7/16]

hex

hex(x)는 정수값을 입력받아 16진수(hexadecimal)로 변환하여 리턴하는 함수이다.

```
>>> hex(234)
'0xea'
>>> hex(3)
'0x3'
```

id

id(object)는 객체를 입력받아 객체의 고유 주소값(레퍼런스)을 리턴하는 함수이다.

```
>>> a = 3
>>> id(3)
135072304
>>> id(a)
135072304
>>> b = a
>>> id(b)
135072304
```

위 예의 3, a, b는 고유 주소값이 모두 135072304이다. 즉, 3, a, b가 모두 같은 객체를 가리키고 있음을 알 수 있다.

int

`int(x)`는 문자열 형태의 숫자나 소수점이 있는 숫자 등을 정수 형태로 리턴하는 함수로, 정수를 입력으로 받으면 그대로 리턴한다.

```
>>> int('3')
3
>>> int(3.4)
3
```

`int(x, radix)`는 `radix` 진수로 표현된 문자열 `x`를 10진수로 변환하여 리턴한다.

isinstance

`isinstance(object, class)`는 첫 번째 인수로 인스턴스, 두 번째 인수로 클래스 이름을 받는다. 입력으로 받은 인스턴스가 그 클래스의 인스턴스인지를 판단하여 참이면 `True`, 거짓이면 `False`를 리턴한다.

```
>>> class Person: pass
...
>>> a = Person()
>>> isinstance(a, Person)
True
```

위의 예는 `a`가 `Person` 클래스에 의해서 생성된 인스턴스임을 확인시켜 준다.

```
>>> b = 3
>>> isinstance(b, Person)
False
```

`b`는 `Person` 클래스에 의해 생성된 인스턴스가 아니므로 `False`를 리턴한다.

Python Built-in Functions [9/16]

lambda

lambda는 함수를 생성할 때 사용하는 예약어로, def와 동일한 역할을 한다. 보통 함수를 한줄로 간결하게 만들 때 사용한다. 우리말로 "람다"라고 읽고 def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰인다. 사용법은 다음과 같다.

lambda 인수1, 인수2, ... : 인수를 이용한 표현식

```
>>> sum = lambda a, b: a+b
>>> sum(3,4)
7
```

lambda를 이용한 sum 함수는 인수로 a, b를 받아 서로 더한 값을 돌려준다. 위의 예제는 def를 사용한 아래 함수와 하는 일이 완전히 동일하다.

```
>>> def sum(a, b):
...     return a+b
...
>>>
```

그렇다면 def가 있는데 왜 lambda라는 것이 나오게 되었을까? 이유는 간단하다. lambda는 def 보다 간결하게 사용할 수 있기 때문이다. 또한 lambda는 def를 사용할 수 없는 곳에도 사용할 수 있다. 다음 예제에서 리스트 내에 lambda가 들어간 경우를 살펴보자.

```
>>> myList = [lambda a,b:a+b, lambda a,b:a*b]
>>> myList
[at 0x811eb2c>, at 0x811eb64>]
```

```
>>> myList[0]
at 0x811eb2c>
>>> myList[0](3,4)
7
```

```
>>> myList[1](3,4)
12
```

len

len(s)은 입력값 s의 길이(요소의 전체 개수)를 리턴하는 함수이다.

```
>>> len("python")
6
>>> len([1,2,3])
3
>>> len((1, 'a'))
2
```

list

list(s)는 반복 가능한 자료형 s를 입력받아 리스트로 만들어 리턴하는 함수이다.

```
>>> list("python")
['p', 'y', 't', 'h', 'o', 'n']
>>> list((1,2,3))
[1, 2, 3]
```

list 함수에 리스트를 입력으로 주면 똑같은 리스트를 복사하여 돌려준다.

```
>>> a = [1, 2, 3]
>>> b = list(a)
>>> b
[1, 2, 3]
```

map(f, iterable)은 함수(f)와 반복 가능한(iterable) 자료형을 입력으로 받는다. map은 입력받은 자료형의 각 요소가 함수 f에 의해 수행된 결과를 묶어서 리턴하는 함수이다.

```
# two_times.py
def two_times(numberList):
    result = [ ]
    for number in numberList:
        result.append(number*2)
    return result

result = two_times([1, 2, 3, 4])
print(result)
```

two_times 함수는 리스트 요소를 입력받아 각 요소에 2를 곱한 결과값을 돌려준다. 실행 결과는 다음과 같다.

결과값: [2, 4, 6, 8]

위의 예제는 map 함수를 이용하면 다음처럼 바꿀 수 있다.

```
>>> def two_times(x): return x*2
...
>>> list(map(two_times, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

앞의 예는 lambda를 사용하면 다음처럼 간략하게 만들 수 있다.

```
>>> list(map(lambda a: a*2, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

max

`max(iterable)`는 인수로 반복 가능한 자료형을 입력받아 그 최대값을 리턴하는 함수이다.

```
>>> max([1, 2, 3])
3
>>> max("python")
'y'
```

min

`min(iterable)`은 `max` 함수와 반대로, 인수로 반복 가능한 자료형을 입력받아 그 최소값을 리턴하는 함수이다.

```
>>> min([1, 2, 3])
1
>>> min("python")
'h'
```

oct

`oct(x)`는 정수 형태의 숫자를 8진수 문자열로 바꾸어 리턴하는 함수이다.

```
>>> oct(34)
'0o42'
>>> oct(12345)
'0o30071'
```


open

`open(filename, [mode])`은 "파일 이름"과 "읽기 방법"을 입력받아 파일 객체를 리턴하는 함수이다.
읽기 방법(mode)이 생략되면 기본값인 읽기 전용 모드(r)로 파일 객체를 만들어 리턴한다.

mode	설명
w	쓰기 모드로 파일 열기
r	읽기 모드로 파일 열기
a	추가 모드로 파일 열기
b	바이너리 모드로 파일 열기

b는 w, r, a와 함께 사용된다.

```
>>> f = open("binary_file", "rb")
```

위 예의 rb는 "바이너리 읽기 모드"를 의미한다.

아래 예의 `fread`와 `fread2`는 동일한 방법이다.

```
>>> fread = open("read_mode.txt", 'r')  
>>> fread2 = open("read_mode.txt")
```

즉, 모드 부분이 생략되면 기본값으로 읽기 모드인 r을 갖게 된다.

다음은 추가 모드(a)로 파일을 여는 예이다.

```
>>> fappend = open("append_mode.txt", 'a')
```

pow

`pow(x, y)`는 `x`의 `y` 제곱한 결과값을 리턴하는 함수이다.

```
>>> pow(2, 4)
16
>>> pow(3, 3)
27
```

range

`range([start,] stop [,step])`는 `for`문과 함께 자주 사용되는 함수이다. 이 함수는 입력받은 숫자에 해당 되는 범위의 값을 반복 가능한 객체로 만들어 리턴한다.

인수가 2개일 경우

입력으로 주어지는 2개의 인수는 시작 숫자와 끝 숫자를 나타낸다. 단, 끝 숫자는 해당 범위에 포함 되지 않는다는 것에 주의하자.

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
```

인수가 3개일 경우

세 번째 인수는 숫자 사이의 거리를 말한다.

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

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

Python Built-in Functions

[15/16]

sorted

`sorted(iterable)` 함수는 입력값을 정렬한 후 그 결과를 리스트로 리턴하는 함수이다.

```
>>> sorted([3, 1, 2])
[1, 2, 3]
>>> sorted(['a', 'c', 'b'])
['a', 'b', 'c']
>>> sorted("zero")
['e', 'o', 'r', 'z']
>>> sorted((3, 2, 1))
[1, 2, 3]
```

```
>>> a = [3, 1, 2]
>>> result = a.sort()
>>> print(result)
None
>>> a
[1, 2, 3]
```

str

`str(object)`은 문자열 형태로 객체를 변환하여 리턴하는 함수이다.

```
>>> str(3)
'3'
>>> str('hi')
'hi'
>>> str('hi'.upper())
'HI'
```

tuple

`tuple(iterable)`은 반복 가능한 자료형을 입력받아 튜플 형태로 바꾸어 리턴하는 함수이다. 만약 튜플이 입력으로 들어오면 그대로 리턴한다.

```
>>> tuple("abc")
('a', 'b', 'c')
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> tuple((1, 2, 3))
(1, 2, 3)
```

type

`type(object)`은 입력값의 자료형이 무엇인지 알려주는 함수이다.

```
>>> type("abc")
<class 'str'>
>>> type([ ])
<class 'list'>
>>> type(open("test", 'w'))
<class '_io.TextIOWrapper'>
```

zip

`zip(iterable*)` 은 동일한 개수로 이루어진 자료형을 묶어 주는 역할을 하는 함수이다.

```
>>> list(zip([1, 2, 3], [4, 5, 6]))
[(1, 4), (2, 5), (3, 6)]
>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> list(zip("abc", "def"))
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

Python Functions

- Basics of Python Functions
- Python Built-in Functions
- Problem Solving using Python Functions

Compute molecular weight

```
# Compute molecular weight
# Here are basic weights
#   carbon(c): 12.011
#   hydrogen(H): 1.0079
#   oxygen(O): 15.9994
#
# use round(46.0688, 2) ==> 46.07
```

```
def molecular_weight():
    print("Please enter the number of each atom")
    C = input("carbon: ")
    H = input("hydrogen: ")
    O = input("oxygen: ")
    W = C*12.011 + H*1.0079 + O*15.9994
    print ("The molecular weight of C",C,"H",H,"O",O,"is: ", round(W,2) )
```

```
def molecular_weight_correct():
    print("Please enter the number of each atom")
    C = eval(input("carbon: "))
    H = eval(input("hydrogen: "))
    O = eval(input("oxygen: "))
    W = C*12.011 + H*1.0079 + O*15.9994
    print ("The molecular weight of C",C,"H",H,"O",O,"is: ", round(W,2))
```

Palindrome Checker [1/2]

- # Palindrome이란 철자를 거꾸로 놓아도 원래와 같은 글귀를 말합니다.
- # 부호와 빈칸을 제외하고 대소문자 구분없이 알파벳이 대칭을 이루는 문장
- # 예를 들어, 'abcdcba'는 뒤집어도 똑같으므로 palindrome
- #
- # 조금 더 복잡한 palindrome 예제들
- # 'Are we not drawn onward, we few, drawn onward to new era'
- # 'Do geese see God'
- # 'Dennis and Edna sinned'

Palindrome Checker [2/2]

```
def pallindrome_decider():
    Pallindrome_candidate = input("Type your pallindrome candiate: ")
    print ("Here is your pallindrome candiate:", Pallindrome_candidate)
    Pallindrome_candidate = Pallindrome_candidate.lower()
    print ("After lowering characters ==> ", Pallindrome_candidate)
    #
    isPallindrome_candidate = True
    p1 = 0
    p2 = len(Pallindrome_candidate)-1
    #
    while isPallindrome_candidate and p1 < p2:
        if Pallindrome_candidate[p1].isalpha():
            if Pallindrome_candidate[p2].isalpha():
                if Pallindrome_candidate[p1]==Pallindrome_candidate[p2]:
                    p1 = p1+1
                    p2 = p2-1
                else: isPallindrome_candidate = False
            else: p2 = p2-1 # if not alphet ==> move p2 to left
        else: p1 = p1+1 # if not alphet ==> move p1 to right
    #
    if isPallindrome_candidate:
        print ("Yes, your pallindrome candiate", Pallindrome_candidate, "is a real pallindrome!")
    else: print ("No, your pallindrome candiate", Pallindrome_candidate, "is not a real pallindrome!")
```


Happy Birth Day Song

```
# sing("Kang Min")  
#  
# Happy birthday to you!  
# Happy birthday to you!  
# Happy birthday, dear Kang Min.  
# Happy birthday to you!  
  
def sing(name):  
    print("Happy birthday to you! ")  
    print("Happy birthday to you! ")  
    print("Happy birthday, dear %s " %name)  
    print("Happy birthday to you! ")
```

Euclidean Distance Computation

Euclidean Distance란 직교 좌표계에서 두 점의 거리를 나타냅니다.
예를 들어, 2차원 평면에서 두 점 (x_1, y_1) , (x_2, y_2) 의 거리는
$\text{math.sqrt}((x_1 - x_2)^2 + (y_1 - y_2)^2)$ 로 계산
이와 같이 임의의 차원에서의 거리를 구하는 함수를 구현해보세요.

함수가 받을 parameter는 총 3개로,
첫번째 parameter n은 차원 수, parameter p1, parameter p2는 길이 가 n인 리스트

```
import math
```

```
def eucDist(n, p1, p2):  
    distance = 0  
    for i in range(n):  
        distance = distance + (p2[i]-p1[i])**2  
    #  
    return math.sqrt(distance)
```

```
>> p1 = [2.0 4.0 6.0]  
>> P2 = [1.5 4.5 10.2]  
>> eucDist(3, p1, p2)
```

Math module

- This module provides access to mathematical functions defined in C standard
 - These functions cannot be used with complex numbers (Use cmath)
- These are the categories of the functions
 - Number-theoretic
 - Power and logarithmic
 - Trigonometric
 - Angular
 - Hyperbolic
 - Constant

Math module – Functions [1/3]

- `math.ceil(x)` : Return the smallest integer greater than or equal to x
- `math.floor(x)` : Return the largest integer less than or equal to x
- `math.fabs(x)` : Return the absolute value of x
- `math.factorial(x)` : Return the x factorial
- `math.fmod(x,y)` : Return the remainder of x divided by y

```
import math

a = -3.123
b = 8
c = 3

print("ceil of a : ", math.ceil(a))
print("floor of a : ", math.floor(a))
print("fabs of a : ", math.fabs(a))
print("factorial of b : ", math.factorial(b))
print("fmod of b,c : ", math.fmod(b,c))
```

Result

```
>>>
ceil of a :  -3
floor of a :  -4
fabs of a :   3.123
factorial of b :  40320
fmod of b,c :   2.0
```

Math module – Functions [2/3]

- `math.log(a, b)` : Return the value of $\log_b a$
- `math.pow(a, b)` : Return the a raised to the power of b
- `math.sqrt(a)` : Return the square root of a
- `math.sin(x)` : Return the sine of x radians
- `math.cos(x)` : Return the cosine of x radians
- `math.tan(x)` : Return the tangent of x radians

```
import math
```

```
a = 2  
b = 4  
c = 25
```

```
print("log a b : ", math.log(b, a))  
print("pow of a,b : ", math.pow(a,b))  
print("sqrt of a : ", math.sqrt(c))  
print("sin(pi) : ", math.sin( math.pi ))  
print("cos(pi) : ", math.cos( math.pi ))  
print("tan(pi) : ", math.tan( math.pi ))
```

Result

```
>>>  
log a b :  2.0  
pow of a,b :  16.0  
sqrt of a :  5.0  
sin(pi) :  1.2246467991473532e-16  
cos(pi) :  -1.0  
tan(pi) :  -1.2246467991473532e-16
```

Close to 0



Math module – Functions [3/3]

- `math.degrees(x)` : Convert angle x from radians to degrees
- `math.radians(x)` : Convert angle x from degrees to radians
- `math.cosh(x)` : Return the hyperbolic cosine of x
- `math.sinh(x)` : Return the hyperbolic sine of x
- `math.tanh(x)` : Return the hyperbolic tangent of x

Result

```
import math

print("degrees of pi : ", math.degrees(math.pi))
print("radians of 180 : ", math.radians(180))
print("cosh of pi : ", math.cosh( math.pi ))
print("sinh of pi : ", math.sinh( math.pi ))
print("tanh of pi : ", math.tanh( math.pi ))
```

```
>>>
degrees of pi : 180.0
radians of 180 : 3.141592653589793
cosh of pi : 11.548739357257746
sinh of pi : 11.591953275521519
tanh of pi : 0.99627207622075
```

Temperature Warning

```
# input은 '20.3F' '-10C' '32.5C' 같은방식의 string으로 입력
# output은
# 물의 끓는 점 이상일 경우 Be careful!
# 물이 어는 점 이하일 경우 Don't get frozen!
# 물 밀도가 가장 높은 점(섭씨 3도에서 5도 사이로 가정)일 경우 You will be fine!
```

```
def FtoC(F):
    C = (F-32)*5/9
    return C
```

```
def TempOK(C):
    if C >= 100:          print ("Be careful!")
    if C <= 0:            print ("Don't get frozen!")
    if C >= 3 and C <= 5: print ("You will be fine")
```

```
def WeatherMessage():
    temp = input("Type your temperature in string format:")
    if temp[-1] == "C":
        Centi = float(temp[:-1])
        TempOK(Centi)
    elif temp[-1] == "F":
        Fahren = float(temp[:-1])
        TempOK(FtoC(Fahren))
    else: print("Pardon?")
```

Leap Year Checker

```
# 윤년은 1년이 366일로 이루어져 있는 해인데
# 규칙은 다음 Wolfram.com에서 주어진 정의와 같습니다.
# Leap years were therefore 45 BC, 42 BC, 39 BC, 36 BC, 33 BC,
# 30 BC, 27 BC, 24 BC, 21 BC, 18 BC, 15 BC, 12 BC, 9 BC, 8 AD,
# 12 AD, and every fourth year thereafter (Tøndering), until the
# Gregorian calendar was introduced (resulting in skipping three out
# of every four centuries).
```

```
def yun_year_checker():
    target_year = input("Please type your year:")
    yun_year = False
    #
    if target_year in [-45, -42, -39, -33, -30, -27, -24, -21, -18, -15, -12, -9, 8, 12]:
        yun_year = True
    #
    elif target_year > 12 and target_year % 4==0:
        yun_year = True
        if target_year % 100==0:
            yun_year = False
            if target_year % 400==0: yun_year = True
    #
    if yun_year: print ("Yes, the year", target_year, " is a leap year!")
    else:        print ("No, the year", target_year, " is not a leap year!")
```


Valid Date Checker [1/2]

입력된 날짜가 유효할 경우 valid, 입력된 날짜가 유효하지 않거나 입력된 값이 날짜
형태가 아닐 경우 invalid을 출력합니다. 유효한 날짜는 달력 상 존재하는 날짜를
의미합니다. 예를 들어, -5/12/17은 기원전 5년의 12월 17일을 의미하므로 유효합니다.
하지만 0년은 존재하지 않습니다.

```
def LeapYear(y):  
    year = y  
    yun = False  
    if year in [-45,-42,-39,-33,-30,-27,-24,-21,-18,-15,-12,-9,8,12]:  
        yun = True  
    elif year > 12 and year%4==0:  
        yun = True  
        if year%100==0:  
            yun = False  
            if year%400==0: yun = True  
    return yun
```

```
def MonthDate(y, m):  
    if m in [1, 3, 5, 7, 8, 10, 12]:  
        return 31  
    elif m == 2:  
        if LeapYear(y): return 29  
        else: return 28  
    else:  
        return 30
```

Valid Date Checker [2/2]

```
def valid_date_checker():
    Target_Date = input("Type your date in yyyy/mm/dd string format:")
    print ("Your Target Date is:", Target_Date)

    try:
        date = Target_Date.split("/")
        print ("Your typed date is:", "Year", date[0], "Month", date[1], "Day", date[2])
        if int( date[0] ) ==0:
            print ("Your typed date is invalid")
        elif int( date[1] ) in [1,2,3,4,5,6,7,8,9,10,11,12]:
            daylist = []
            for i in range( MonthDate(int(date[0]), int(date[1])) ):
                daylist.append( i+1 )
            if int(date[2]) in daylist: print ("Your typed date is valid!")
            else: print ("Your typed date is invalid!")
        else:
            print ("Your typed date is invalid")

    except:
        print ("Your typed date is invalid")
```