Functional arguments passing types
- Default argument
- Required argument
- Keyword argument
- Variable length argument

In [10]:

```
# Default argument

def sa(p,m=3):
    print(p,m)
    return

n = int(input())
l = int(input())
sa(n)
```

```
56
67
56 3
```

In [21]:

```
# Required argument (or) Positional argument

def sd(p,q,l=90):
    print(p,q,l)
    return


n = input()
m = input()
k = input()
sd(n,m,k)
# sd(45,12)
```

```
23
13
15
23 13 15
```

In [32]:

```
# Keyword argument

def na(n,p,ls=89):
    print(n,p,ls)
    return

n = int(input())
ls = int(input())
p = int(input())
na(ls='90',n='56',p='85')
```

```
1
3
2
56 85 90
```

In [39]:

```
# Variable Length argument

def maru(e,*er):
    print(e,end="\n")
    for i in er:
        if i%2 == 0:
            print(i,end=" ")
#     print(er,type(er))
    return

maru(45,78,23,90,34,5,6,7,8,12,89)
```

```
45
78 90 34 6 8 12
```

Python Data Structures

- String -> '' or " "
- List -> any type of data
- Tuple -> any type of data
- Set -> any type of data
- Dictionary -> (key,value) pairs

Strings

In [44]:

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

In [45]:

```
print(dir(str))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc_
_', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__
getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__ite
r__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new_
_', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__se
tattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefo
ld', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'forma
t', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'is
digit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'i
stitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partit
ion', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstri
p', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'tran
slate', 'upper', 'zfill']
```

In [48]:

```
help(str.capitalize)
```

```
Help on method_descriptor:

capitalize(self, /)
    Return a capitalized version of the string.

    More specifically, make the first character have upper case and the rest
lower
    case.
```

In [49]:

```
m = "raju"
```

In [50]:

```
m
```

Out[50]:

```
'raju'
```

In [51]:

```
1 m.capitalize()
```

Out[51]:

```
'Raju'
```

In [ ]:

```
1
```