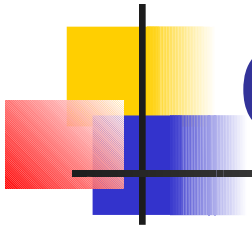
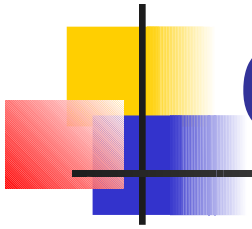


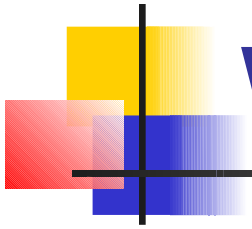
CS105

FUNCTION



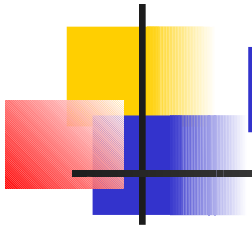
Overview

- What is function?
- Definition & Calling of function
- Parameter passing
- return statement
- Dynamic data typing
- Scoping rule



What is function?

- Collection of statements
- Repeating execution
- Encapsulation
- Generating new operation
- Separation of code into logical unit
- Design of hierarchical programming



Definition & calling of func

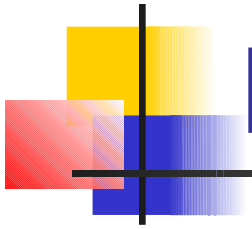
- Definition

```
def funcName(parmeterList):
    statements
    return returnValue
```

```
>>> def add(a,b):
                    return a+b
```

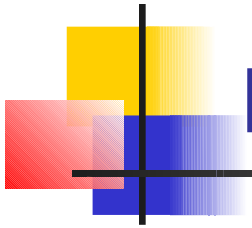
- Calling

```
>>> add
<function add at XXXXXX>
>>> c = add(1,2)
>>> c
3
>>> d = add
>>> d(4,5)
9
```



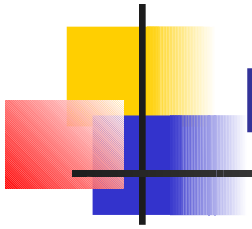
Parameter passing

```
>>> def myFunc(p):
        r = 100
>>> a = 200
>>> myFunc(a)
>>> print a
200
```



return statement

```
>>> def doNoting():
        return
>>> doNothing()
>>> a = doNothing()
>>> print a
None

>>> def doSimple():
        pass
>>> doSimple()
>>> print doSimple()
None
>>>
```



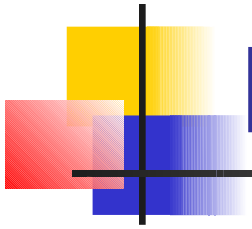
return statement (cont.)

- Return one value

```
>>> def  myAbs(x)
    if x < 0:  return –x
    return x
```

- Return multiple values

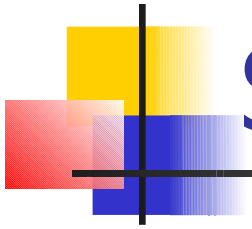
```
>>> def mySwap(x,y):
    return y,x

>>> a = 100
>>> b = 200
>>> a, b = mySwap(a, b)
>>> x = mySwap(a, b)
```



Dynamic data typing

- Operation of object is defined at execution time
- Dynamically determined

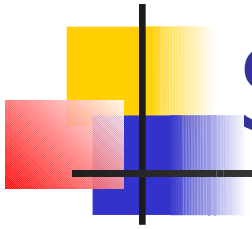
```
>>> def myAdd(a, b):
    return a+ b

>>> c = myAdd(1, 2.3)
>>> d = myAdd('data', 'type')
>>> e = myAdd(['operation of object'],
['is determined', 'at execution time'])
```



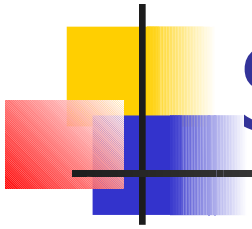
Scoping rule

- name space
  - Space where names (variables, functions, classes, instances, and so on) are defined
- Types of name spaces
  - local
    - In function
  - Global
    - In module (i.e., file)
  - Built-in

Scoping rule (cont.)

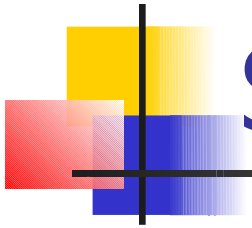


- LGB rule
  - Built-in c=5, d=6, e=7
  - Global b=3, c=4
  - Local a=1, b=2



Scoping rule (cont.)

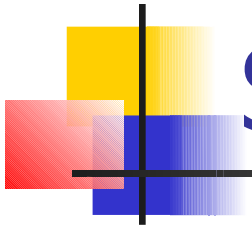
```
>>> d = 4
>>> def func(a, b):
                c = a + b
                print a, b, c, d, __builtins__
>>> func(1, 2)
1 2 3 4 <module '__builtin__' (built-in)>
>>> abs
<built-in function abs>
```



Scoping rule (cont.)

- Variable is created in the name space where it is defined

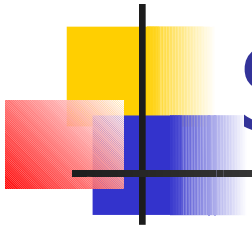
```
a = 100
b = 200

def myFunc(c):
    d = c + 100
    e = c + b
    return e
```



Scoping rule (cont.)

- Problem occurs when global variables are used in local name space
  - UnboundLocalError

```
a = 100

def myFunc():
        b = a
        a = 200
        return b

myFunc()
```



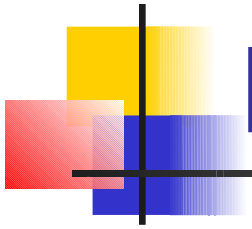
Scoping rule (cont.)

- Use global to resolve unboundLocalError

```
def myFunc():
    global a
    b = a
    a = 200
    return b
```

- How to list built-ins

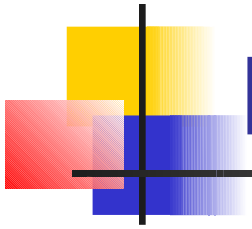
```
>>> dir(__builtins__)
```



Function parameter

- Default parameter value
  - NOTE: default parameters should be positioned after non-default parameters

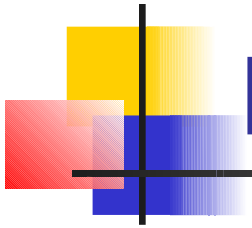
```
>>> def myIncrease(a, step=1):
      return a + step

>>> b = 1
>>> b = myIncrease(b)
>>> c = myIncrease(b, 100)
>>> b, c
```



Function parameter (cont.)

- Keyword parameter
  - Passing through parameter name

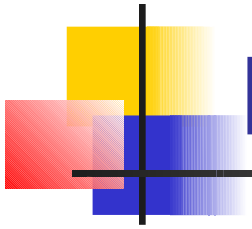
```
>>> def myArea(height, width):
        return height * width


>>> a = area(width=100, height=200)
>>> b = area(height='height ', width=3)
>>> c = area(300, width=400)
```



Function parameter (cont.)

- Variable length parameter list
  - First, fixed paramaters are passed
  - After then, variable length parameters are passed in a tuple

```
>>> def myVLP(a, *vlp):
        print a, vlp

>>> myVLP(100)
100 ()
>>> myVLP(100, 200)
100 (200, )
>>> myVLP(100, 200, 300, 400, 500)
100 (200, 300, 400, 500)
```



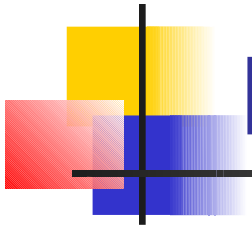
Function parameter (cont.)

- Variable length parameter
  - For example, printf() in C

    printf("I've spent %d days and %d nights to do this.", 2, 1)

- Let's make printf() in Python

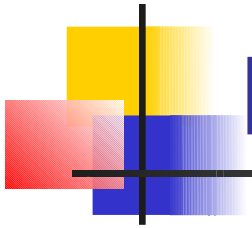
```
def printf(format, *args):
    print format % args
```



Function parameter (cont.)

- Undefined keyword parameter
  - Should be positioned at the end of parameter list with **

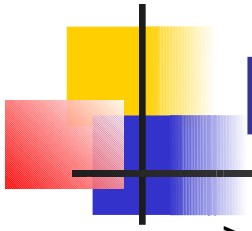
```
>>> def myUKP(width, height, **ukp):
        print width, height
        print ukp

>>> myUKP(width=100, height=200, depth=300, dimension=400)
100 200
{'depth': 300, 'dimension':400}
```



Function parameter (cont.)

- Order of function parameters
  - First, usual parameters are passed in order
  - Second, variable length parameters are passed in tuple
  - Third, keyword parameters are passed in dictionary

```
>>> def myParameter(a, b, *vlp, **ukp)
                print a, b
                print vlp
                print ukp

>>>  myParameter(100, 200, 300, 400, 500, c=600, d=700)
100 200
(300, 400, 500)
{'c':500, 'd':700}
```



Function parameters (cont.)

```
>>> def myFunc(a,b,c)
        print a, b, c
```

- Calling a function using tuple

  ```
  >>> targs = (100, 200, 300)
  >>> myFunc(*args)
  100 200 300
  ```

- Calling a function using dictionary

  ```
  >>> dargs = {'a':100, 'b':200, 'c':300}
  >>> myFunc(**dargs)
  100 200 300
  ```

- Calling a function using both tuple and dictionay

  ```
  >>> targs = (100, 200, 300)
  >>> dargs = {'a':100, 'b':200, 'c':300}
  >>> functionName(*targs, **dargs)
  ```