III B.Tech.

Computer Science & Engineering

CSE304: PYTHON PROGRAMMING WITH WEB FRAMEWORKS

Extended Sequence Assignments, Packing and Unpacking, Arguments Matching Modes

By

Mrs. S. KAMAKSHI, AP-III / CSE

School of Computing

Extended Sequence Assignments

```
t = (1,2,3,4)
a, b, c, d = t                        # a= 1 b= 2 c= 3 d= 4
a, b =t                               # ValueError Exception
a, *b = t                             # a= 1 b= [2, 3, 4]
*a, b = t                             # a= [1, 2, 3] b= 4
a, *b, c, d = t                       # a= 1 b= [2] c= 3 d= 4
a, b, *c, d, e = t                    # a= 1 b= 2 c= [] d= 3 e= 4
a, *b, c = {10, 20, 30, 40, 50, 60}
                                      # a= 40 b= [10, 50, 20, 60] c= 30
a, b = {12:45, 6:'sss'}               # a= 12 b= 6
a, *b, c, *d = [1, 2, 3, 4, 5, 6]
        #SyntaxError: Two starred expressions not allowed in an
        assignment
```

Extended Sequence in Loops

```
L = [1, 2, 3, 4, 5]
while L:
    first, *L = L
    print(first, L)
```

Output:

```
1 [2, 3, 4, 5]
2 [3, 4, 5]
3 [4, 5]
4 [5]
5 []
[]
```

```
for (a, *b, c) in [(1,2,3,4),   (5,6,7), (8, 9, 10, 11, 12)]:
     print (‘a=‘, a, ‘b=‘, b, ‘c=‘,c)
```

Output:

```
a= 1 b= [2, 3] c= 4
a= 5 b= [6] c= 7
a= 8 b= [9, 10, 11] c= 12
```

Shared References vs. Mutable and Immutable Objects

- L = [1, 2]
- M = L
- L = L + [3, 4]

- T = (1, 2)
- M = T
- T = T + (3, 4)

- L = [1, 2]
- M = L
- L += [3, 4]

- T = (1, 2)
- M = T
- T += (3, 4)

Passing Arguments to Functions

- Arguments are passed by automatically assigning objects to local variable names.
- Immutable arguments are effectively passed "by value."
  - Assigning to argument names inside a function does not affect the caller.
- Mutable arguments are effectively passed "by reference."
  - Changing a mutable object argument in a function may impact the caller

Argument Matching Modes

- **Positional: matched from left to right**
  - match passed argument values to argument names in a function header by position, from left to right
- **Keywords: matched by argument name**
  - Callers can specify which argument in the function is to receive a value by using the argument's name in the call, with the name=value syntax.
- **Defaults: specify values for optional arguments that aren't passed**
  - Functions themselves can specify default values for arguments to receive if the call passes too few values, again using the name=value syntax.
- **Varargs collecting: collect arbitrarily many positional or keyword arguments**
  - Functions can use special arguments preceded with one or two * characters to collect an arbitrary number of possibly extra arguments.
- **Varargs unpacking: pass arbitrarily many positional or keyword arguments**
  - Callers can also use the * syntax to unpack argument collections into separate arguments. This is the inverse of a * in a function header—in the header it means collect arbitrarily many arguments, while in the call it means unpack arbitrarily many arguments, and pass them individually as discrete values.
- **Keyword-only arguments: arguments that must be passed by name**
  - Functions can also specify arguments that must be passed by name with keyword arguments, not by position. Such arguments are typically used to define configuration options in addition to actual arguments.

Argument Matching Forms

Syntax	where	Interpretation
fun_name(value)	caller	Normal Argument, Matched by position
fun_name(name=value)	caller	Keyword Argument, Matched by name
fun_name(*iterable)	caller	Pass all objects in iterable as individual positional arguments
fun_name(**dict)	caller	Pass all key/value pairs in dictionary as individual keyword arguments
def fun_name(name)	header	Normal Argument, Matches any passed value by position or name
def fun_name(name=value)	header	Default argument value, if not passed in the call
def fun_name(*name)	header	Matches and collects remaining positional arguments in a tuple
def fun_name(**name)	header	Matches and collects remaining keyword arguments in a dictionary
def fun_name(*other, name)	header	Arguments that must be passed by keyword only in call
def fun_name(*, name=value)	header	

Order of Arguments

- In a function call, arguments must appear in this order:
  - positional arguments (value);
  - keyword arguments (name=value) ;
  - *iterable form;
  - **dict form
- In a function header, arguments must appear in this order:
  - normal arguments (name);
  - default arguments (name=value);
  - *name form;
  - name or name=value keyword-only arguments;
  - **name form.
- Order of Matching Arguments
  - Assign non-keyword arguments by position.
  - Assign keyword arguments by matching names.
  - Assign extra non-keyword arguments to *name tuple.
  - Assign extra keyword arguments to **name dictionary.
  - Assign default values to unassigned arguments in header.