BIOS6642

Tuples and Sets

Fuyong Xing

Department of Biostatistics and Informatics

Colorado School of Public Health

University of Colorado Anschutz Medical Campus

Outline

- **Tuples**
- Tuples and Lists
- Variable-length arguments
- Sets

Tuples

- A tuple is a sequence of values separated by commas.
- The following two statements are equivalent: both create a tuple, *t.*
- The parentheses are optional.

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Tuples

- Create an empty tuple

```
>>> t = tuple()
>>> t
()
```

- Create a tuple with a single element

```
>>> t1 = 'a',
>>> type(t1)
<class 'tuple'>
```

Note there is a comma.

- How about *t1=('a')*?

Tuples

- Convert a string to a tuple

```
>>> t = tuple('lupins')
>>> t
('l', 'u', 'p', 'i', 'n', 's')
```

- How about

```
>>> tuple([8, 'a', '2'])
```

Tuples

- Similar to lists, we can apply indexing and slicing to tuples.
- Note that tuples are immutable.

Feature	List	Tuple
Mutability	mutable	immutable
Creation	`lst = [i, j]`	`tpl = (i, j)`
Element access	`a = lst[i]`	`a = tpl[i]`
Element modification	`lst[i] = a`	*Not possible*
Element addition	`lst += [a]`	*Not possible*
Element removal	`del lst[i]`	*Not possible*
Slicing	`lst[i:j:k]`	`tpl[i:j:k]`
Slice assignment	`lst[i:j] = []`	*Not possible*
Iteration	`for elem in lst:...`	`for elem in tpl:...`

Codes

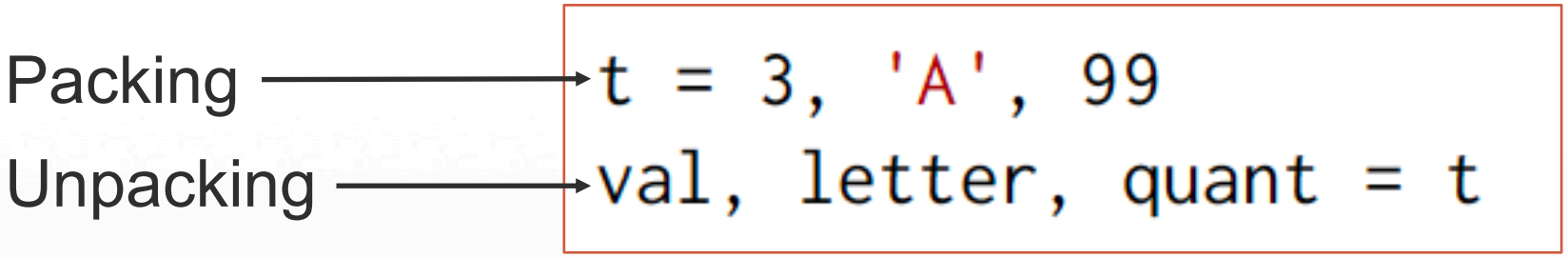
```
my_list = [1, 2, 3, 4, 5, 6, 7]     # Make a list
my_tuple = (1, 2, 3, 4, 5, 6, 7)    # Make a tuple
print('The list:', my_list)          # Print the list
print('The tuple:', my_tuple)        # Print the tuple
print('The first element in the list:', my_list[0])   # Access an element
print('The first element in the tuple:', my_tuple[0]) # Access an element
print('All the elements in the list:', end=' ')
for elem in my_list:                 # Iterate over the elements of a list
    print(elem, end=' ')
print()
print('All the elements in the tuple:', end=' ')
for elem in my_tuple:                 # Iterate over the elements of a tuple
    print(elem, end=' ')
print()
print('List slice:', my_list[2:5])     # Slice a list
print('Tuple slice:', my_tuple[2:5])   # Slice a tuple
print('Try to modify the first element in the list . . .')
my_list[0] = 9           # Modify the list
print('The list:', my_list)
print('Try to modify the first element in the list . . .')
my_tuple[0] = 9          # Is tuple modification possible?
print('The tuple:', my_tuple)
```

Output

```
The list: [1, 2, 3, 4, 5, 6, 7]
The tuple: (1, 2, 3, 4, 5, 6, 7)
The first element in the list: 1
The first element in the tuple: 1
All the elements in the list: 1 2 3 4 5 6 7
All the elements in the tuple: 1 2 3 4 5 6 7
List slice: [3, 4, 5]
Tuple slice: (3, 4, 5)
Try to modify the first element in the list . . .
The list: [9, 2, 3, 4, 5, 6, 7]
Try to modify the first element in the list . . .
Traceback (most recent call last):
  File "tupletest.py", line 26, in <module>
    main()
  File "tupletest.py", line 22, in main
    my_tuple[0] = 9
TypeError: 'tuple' object does not support item assignment
```

Tuples

- Tuple assignment
  - Assign the values of *3, 'A',* and *99* to the variables of *val, letter* and *quant*, respectively.



- The number of variables on the left side are equal to the number of values on the right side.

Tuples

- The unpacking also works when the right side is a string or list.

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
>>> uname
'monty'
>>> domain
'python.org'
```

This method returns a list.

Tuples

- Tuples as return values

Codes

```
def min_max(t):
    return min(t), max(t)

my_tuple = 9, -2, 10, 3, -32, 98
min_max_values = min_max(my_tuple)
min_value, max_value = min_max(my_tuple)

print('The tuple is ', my_tuple)
print('min = {0}, max = {1}'.format(min_max_values[0], min_max_values[1]))
print('min = ', min_value, ', max = ', max_value, sep = '')
```

Output

```
The tuple is  (9, -2, 10, 3, -32, 98)
min = -32, max = 98
min = -32, max = 98
```

Outline

- Tuples
- **Tuples and Lists**
- Variable-length arguments
- Sets

Tuples and lists

- Conversions between tuples and lists

```
>>> tpl = 1, 2, 3, 4, 5, 6, 7, 8
>>> tpl
(1, 2, 3, 4, 5, 6, 7, 8)
>>> list(tpl)
[1, 2, 3, 4, 5, 6, 7, 8]
>>> lst = ['a', 'b', 'c', 'd']
>>> lst
['a', 'b', 'c', 'd']
>>> tuple(lst)
('a', 'b', 'c', 'd')
```

Tuples and lists

- The *zip* function takes two or more sequences and interleaves them. It returns an object that iterates a sequence.

```
>>> lst1 = [1, 2, 3, 4, 5, 6, 7, 8]
>>> lst2 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
>>> for t in zip(lst1, lst2):
...     print(t)
...
(1, 'a')
(2, 'b')
(3, 'c')
(4, 'd')
(5, 'e')
(6, 'f')
(7, 'g')
(8, 'h')
```

Tuples and lists

- The *zip* function does not return a list.
- If the sequences to be paired are not the same length, the result has the length of the shorter one.

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

```
>>> list(zip('Anne', 'Elk'))
[('A', 'E'), ('n', 'l'), ('n', 'k')]
```

Tuples and lists

- Print sum of the components of each tuple

```
>>> for (x, y) in zip([1, 2, 3, 4, 5], [10, 11, 12, 13, 14]):
...     print(x + y)
...
11
13
15
17
19
```

Tuples and lists

- Print sum of the components of each tuple

```
>>> for (x, y) in zip([1, 2, 3, 4, 5], [10, 11, 12, 13, 14]):
...     print(x + y)
...
11
13
15
17
19
```

- Use list comprehensions

```
>>> [x + y for (x, y) in zip([1, 2, 3, 4, 5], [10, 11, 12, 13, 14])]
[11, 13, 15, 17, 19]
```

Outline

- Tuples
- Tuples and Lists
- **Variable-length arguments**
- Sets

Variable-length arguments

- A function adding two numbers

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

- With default parameters, the function can add up to 5 numbers:

```
def sum(a, b=0, c=0, d=0, e=0):
    return a + b + c + d + e
```

Variable-length arguments

- A function adding two numbers

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

- With default parameters, the function can add up to 5 numbers.

```
def sum(a, b=0, c=0, d=0, e=0):
    return a + b + c + d + e
```

- How about adding 1000 numbers?

Variable-length arguments

- A function can take a variable number of arguments by using the asterisk, ∗.
- A parameter that begins with a ∗ gathers arguments into a tuple.

Codes

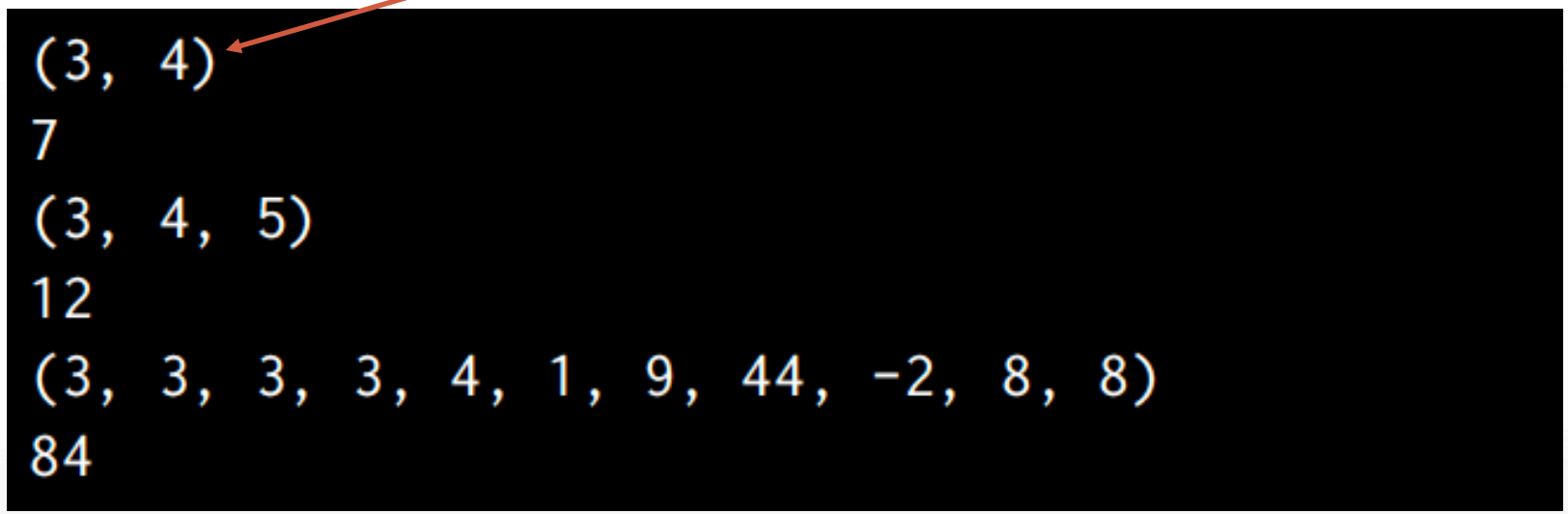
```
def sum(*nums):
    print(nums)          # See what nums really is
    s = 0                # Initialize sum to zero
    for num in nums:     # Consider each argument passed to the function
        s += num         # Accumulate their values
    return s             # Return the sum

print(sum(3, 4))
print(sum(3, 4, 5))
print(sum(3, 3, 3, 3, 4, 1, 9, 44, -2, 8, 8))
```

Variable-length arguments

- A function can take a variable number of arguments by using the asterisk, ∗.
- A parameter that begins with a ∗ gathers arguments into a tuple.

nums is a tuple.

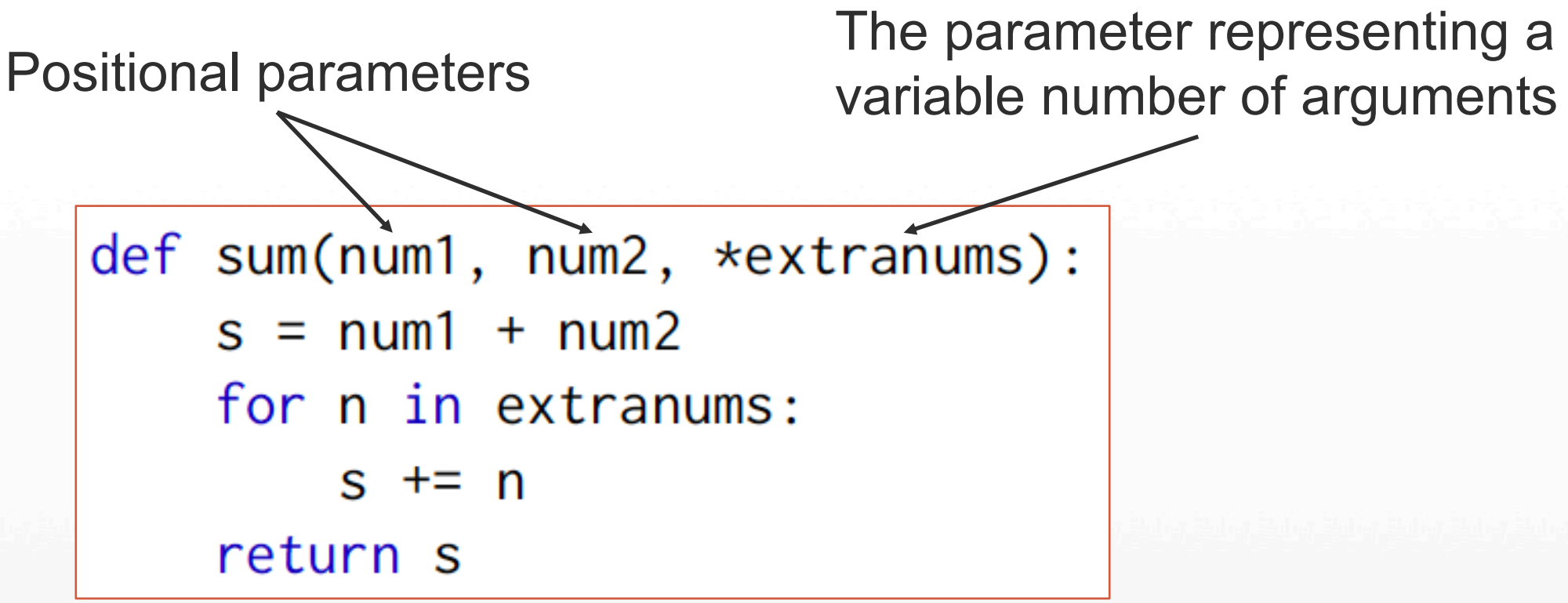


Output

```
(3, 4)
7
(3, 4, 5)
12
(3, 3, 3, 3, 4, 1, 9, 44, -2, 8, 8)
84
```

Variable-length arguments

- If a function has a parameter, which represents a variable number of arguments, and regular positional parameters, then this parameter should appear after those positional parameters.



Variable-length arguments

- Pass a single argument to a function that accepts multiple parameters

Codes

```
def f(a, b, c, d):
    print('a =', a, ' b = ', b, ' c = ', c, ' d = ', d)

args = (10, 20, 30, 40)
f(*args)
```

Output

```
a = 10   b =  20   c =  30   d =  40
```

Outline

- Tuples
- Tuples and Lists
- Variable-length arguments
- **Sets**

Sets

- A set is an unordered collection with no duplicate elements.
- Curly braces, *{}*, are used to enclose the elements of a literal set.

```
>>> S = {10, 3, 7, 2, 11}
>>> S
{2, 11, 3, 10, 7}
>>> T = {5, 4, 5, 2, 4, 9}
>>> T
{9, 2, 4, 5}
```

- Note that the expression, *{}*, does not created an empty set. The expression, *set(),* creates an empty set.

Sets

- Set operations

Operation	Mathematical Notation	Python Syntax	Result Type	Meaning
Union	$A \cup B$	`A | B`	`set`	Elements in A or B or both
Intersection	$A \cap B$	`A & B`	`set`	Elements common to both A and B
Set Difference	$A - B$	`A - B`	`set`	Elements in A but not in B
Symmetric Difference	$A \oplus B$	`A ^ B`	`set`	Elements in A or B, but not both
Set Membership	$x \in A$	`x in A`	`bool`	x is a member of A
Set Membership	$x \notin A$	`x not in A`	`bool`	x is not a member of A
Set Equality	$A = B$	`A == B`	`bool`	Sets A and B contain exactly the same elements
Subset	$A \subseteq B$	`A <= B`	`bool`	Every element in set A also is a member of set B
Proper Subset	$A \subset B$	`A < B`	`bool`	A is a subset B, but B contains at least one element not in A

Sets

- Set operations

```
>>> S = {2, 5, 7, 8, 9, 12}
>>> T = {1, 5, 6, 7, 11, 12}
>>> S | T
{1, 2, 5, 6, 7, 8, 9, 11, 12}
>>> S & T
{12, 5, 7}
>>> 7 in S
True
>>> 11 in S
False
```

Sets

- Set comprehensions

```
>>> S = {x**2 for x in range(10)}
>>> S
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

Readings

- Think Python, chapter 12

References

- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman