LECTURE 12: TUPLES

Hung-Yu Wei

Tuple: a kind of sequence

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

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

() is optional

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

, at the end is needed

```
>>> t2 = ('a')
>>> type(t2)
<class 'str'>
```

This is string

Tuple()

- Empty tuple

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

- Convert to a tuple
  - *tuple()*

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

[] to access elements in tuple

- []
- index

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> t[0]
'a'
```

- slice

```
>>> t[1:3]
('b', 'c')
```

Tuple is immutable

- Tuple is **immutable**
  - *TypeError*
  - *Review: list is mutable*

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

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

- What if I want to modify a tuple?
  - *Create a new tuple*

```
>>> t = ('A',) + t[1:]
>>> t
('A', 'b', 'c', 'd', 'e')
```

Relational operators

- >
- <
- >=
- <=
- ==
- !=
- Compare 1 element by 1 element

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

Tuple assignment

- **Swapping** with tuple assignment

```
>>> a, b = b, a
```

- Conventional swapping for 2 variables

```
>>> temp = a
>>> a = b
>>> b = temp
```

Example: tuple assignment

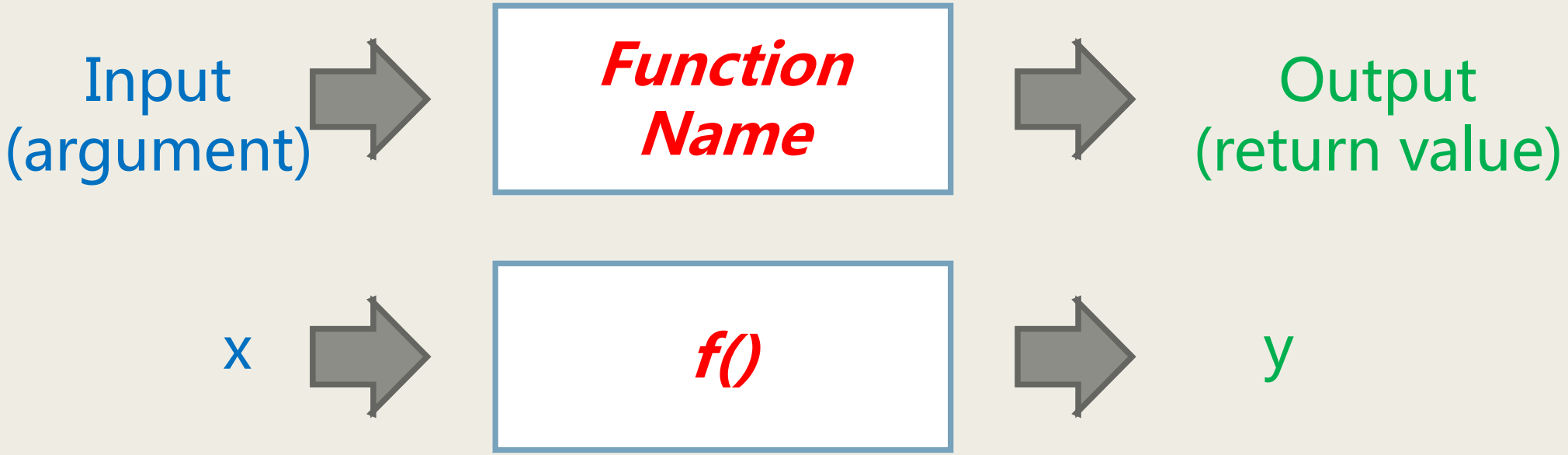
```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

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

```
>>> uname
'monty'
>>> domain
'python.org'
```

Review: Function

$$y = f(x)$$



How do we create a function with multiple return values?

Function with multiple return values

- A function can only have one return
- However, you could return a tuple
  - *Multiple values in a tuple*
- Example: **divmod()**
  - *Division to get quotient and remainder*
- Define your function with 2 return values
  - *Example: min & max*

```
>>> t = divmod(7, 3)
>>> t
(2, 1)
```

```
>>> quot, rem = divmod(7, 3)
>>> quot
2
>>> rem
1
```

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

Variable-length argument tuple with *

- *
  - *Gather*
    - Take any number of argument
  - *Scatter*
    - Input multiple arguments into a function with 1 tuple

```
def printall(*args):
    print(args)
```

```
>>> printall(1, 2.0, '3')
(1, 2.0, '3')
```

```
>>> t = (7, 3)
>>> divmod(t)
TypeError: divmod expected 2 arguments, got 1
```

```
>>> divmod(*t)
(2, 1)
```

Zip object

- Zip a string and a list

```
>>> s = 'abc'
>>> t = [0, 1, 2]
>>> zip(s, t)
<zip object at 0x7f7d0a9e7c48>
```

```
>>> for pair in zip(s, t):
...     print(pair)
...
('a', 0)
('b', 1)
('c', 2)
```

- List and zip

```
>>> list(zip(s, t))
[('a', 0), ('b', 1), ('c', 2)]
```

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

Traverse a list of tuple

```
t = [('a', 0), ('b', 1), ('c', 2)]
for letter, number in t:
    print(number, letter)
```

Traversing 2+ sequences at the same time

- Using, ***for, zip, tuple***

```
def has_match(t1, t2):
    for x, y in zip(t1, t2):
        if x == y:
            return True
    return False
```

Traverse the element and index of a sequence

- Enumerate

```
for index, element in enumerate('abc'):
    print(index, element)
```

```
0 a
1 b
2 c
```

Dictionary and tuple

- item()
  - *Dict_item object that iterates the key-value pairs*

```
>>> d = {'a':0, 'b':1, 'c':2}
>>> t = d.items()
>>> t
dict_items([('c', 2), ('a', 0), ('b', 1)])
```

```
>>> for key, value in d.items():
...     print(key, value)
...
c 2
a 0
b 1
```

More about dictionary and tuple

- Create dictionary with a list of tuples

```
>>> t = [('a', 0), ('c', 2), ('b', 1)]
>>> d = dict(t)
>>> d
{'a': 0, 'c': 2, 'b': 1}
```

- Combine dict with zip to create a dictionary

```
>>> d = dict(zip('abc', range(3)))
>>> d
{'a': 0, 'c': 2, 'b': 1}
```

Example: telephone directory

```
directory[last, first] = number
```

```
for last, first in directory:
    print(first, last, directory[last,first])
```

tuple

0	'Cleese'
1	'John'

dict

('Cleese', 'John')	'08700 100 222'
('Chapman', 'Graham')	'08700 100 222'
('Idle', 'Eric')	'08700 100 222'
('Gilliam', 'Terry')	'08700 100 222'
('Jones', 'Terry')	'08700 100 222'
('Palin', 'Michael')	'08700 100 222'

Sequences of sequences

- Sequence
  - *String, list, tuple*
- String
  - *more limited than other sequences because the elements have to be characters*
  - *Immutable*
- List
  - Lists are more common than tuples, mostly because they are mutable.
- Tuple
  - In some contexts, like a return statement, it is syntactically simpler to create a tuple than a list.
  - If you want to use a sequence as a dictionary key, you have to use an immutable type like a tuple or string.
  - If you are passing a sequence as an argument to a function, using tuples reduces the potential for unexpected behavior due to aliasing.

Sort and reverse

- For mutable sequence
  - *Sort*
  - *Reverse*
- For immutable (e.g. tuple), create a new sequence with
  - *Sorted*
  - *Reversed*

Reading

- Chapter 12 in textbook “Think Python”