Introduction to Python Programming

Thursday, February 12, 2026

1:30pm - 3:30pm **(Online)**

Sherman Centre for Digital Scholarship

```
        self.logger = logging.getLogger(__name__)
        if path:
            self.file = open(os.path.join(
            self.file.seek(0)
            self.fingerprints.update(

    @classmethod
    def from_settings(cls, settings):
        debug = settings.getbool('DUPEFILTER_DEBUG')
        return cls(job_dir(settings), debug)

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)
```

Sherman Centre for Digital Scholarship

Introduction to Python Programming

Presenter: Vivek Jadon, Data Specialist

Date: February 12, 2026

McMaster University | Library

Land Acknowledgement

McMaster University is located on the traditional Territories of the Mississauga and Haudenosaunee Nations, and within the lands protected by the “Dish With One Spoon” wampum agreement.

Code of Conduct

The Sherman Centre and the McMaster University Library are committed to fostering a supportive and inclusive environment for its presenters and participants.

As a participant in this session, you agree to support and help cultivate an experience that is collaborative, respectful, and inclusive, as well as free of harassment, discrimination, and oppression. We reserve the right to remove participants who exhibit harassing, malicious, or persistently disruptive behaviour.

Please refer to our code of conduct webpage for more information:
scds.ca/events/code-of-conduct

Session Recording and Privacy

This session is being recorded with the intention of being shared publicly via the web for future audiences. In respect of your privacy, participant lists will not be shared outside of this session, nor will question or chat transcripts.

Questions asked via the chat box will be read by the facilitator without identifying you. Note that you may be identifiable when asking a question during the session in an audio or visual format.

Book an Appointment with the DASH Team

Receive help from a member of the DASH team! DASH can assist with the following topics:

- Creating data visualizations, including charts, graphs, and scatter plots
- Figuring out which statistical tests to run (e.g., t-test, chi-square, etc.).
- Analyzing data with software including SPSS, Python, R, SAS, ArcGIS, MATLAB, and Excel
- Choosing which software package to use, including free and open-source software
- Troubleshooting problems related to file formats, data retrieval, and download
- Selecting methodology and type of data analysis to use in a thesis project

Book an appointment: **https://library.mcmaster.ca/services/dash**

Certificate Programs

The Sherman Centre for Digital Scholarship Certificate of Attendance

The Sherman Centre's certificate program recognizes attendance at our workshops. It complements degree training, supports the development of critical competencies in data analysis, research data management, and digital scholarship, and formalizes core skills fostered by our workshops.

Participants are invited to attend seven workshops and receive a certificate of attendance. To verify your participation in today's workshop, we will provide a code and additional instructions at the end of the session.

You can learn more about the certificate program at **scds.ca/certificate-program**

The Canadian Certificate for Digital Humanities

This workshop is also eligible for the Canadian Certificate for Digital Humanities. To learn more about the certificate, visit **ccdhhn.ca**. You can also contact local liaison Alexis-Carlota Cochrane at scds@mcmaster.ca

Winter 2026: Upcoming Workshops

Data Analysis Support Hub

March 4: Designing Complex Surveys with REDCap
March 10: Create an Interactive Dashboard using ArcGIS
March 26: Creating Data Visualizations with Javascript
April 2: Microdata Analysis with Python using Statistics Canada Data

Digital Research

February 11: Visualizing Bibliometric Networks with VOSviewer

Research Data Management

February 19: Communities Empowered by Data 101: Tools and Best Practices

May 12: Data Management Plan Bootcamp (In-Person)

May 19: Data Deposit Bootcamp (In-Person)

Do More with Digital Scholarship

February 6: Create a Digital Exhibition with Omeka S

February 9: Rethinking “Good” Data: Power, Vulnerability, and Queer Data Care

Register for Upcoming Workshops: https://u.mcmaster.ca/scds-workshops

Learning Objectives

By the end of this workshop, you will:

- Gain knowledge of the basics of Python programming
- Understand different data types in Python
- Will be able to write simple programs in Python

What is Python?

- Python is a very high-level dynamic object-oriented programming language
- Python is easy to program and read
- Similar to PERL, but with powerful typing and object-oriented features.
- Commonly used for producing HTML content on website
- Useful built-in types (list, dictionaries)
- Clean syntax
- Great for text processing

What is Python?

Invented in the Netherlands in early 90s by Guido van Rossum.

Named after “Monty Python”, a comedy group, as python is fun to use.

Open source and interpreted language.

Considered a scripting language, but it is much more than that.

Scalable, object oriented and functional.

What is Python?

- Python is used by…

YouTube

Dropbox

YAHOO!

Google™

… and many more organizations

Traditional Use of Python

- Image processing
- Embedded scripting
- Artificial Intelligence
- Database programming
- System utilities
- Internet scripting

Python Timeline

Python 1.0 January 1994

Python 2.0 October 2000

Python 3.0 December 2008

Python Integrated Development Environment (IDE)

Any text editing software can be used to write a Python script file. Make sure you save it as .PY file

IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers, etc. to the programmer for application development

Some IDE to consider PyCharm, Spyder, Jupyter, IDLE, Sublime Text, Microsoft Visual Studio Code (VS Code) etc.

Anaconda – Python Distribution

Free and open-source distribution of the **Python** and **R programming** languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

Anaconda bundles a whole bunch of Python packages such as Spyder IDE, Jupyter Notebook, Orange 3, R Studio etc.

Works with Windows, Mac OS and Linux platforms

Jupyter Notebook

McMaster has access to Jupyter notebook via Compute Canada

https://mcmaster.syzygy.ca/

Download Exercise File

- http://bit.ly/2MVaTmv

Data Types in Python

Python has five standard data types:

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

Data Types in Python: Numbers

Python support several different numeric types:

Integers

- Example: 0, 1, 1234, -56
- Dividing an integer by another integer will return a float (to get only the integer part of the quotient use **//** e.g. typing 7//2 will only yield 3)

Long Integers: Only in Python 2; Not in Python 3

- Example: 999999999999999999L
- Must end in either l or L
- Can be arbitrarily long

Floating point numbers

- Example: 0., 1.0, 1e10, 3.14e-2, 6, 99E4
- Division works normally for floating point numbers: 7/2=3.5 (7//2=3)
- Operation involving both floats and integers will yield floats: 6.4-2=4.4, 6.4//2=3.0, 6.4//2.2=2.0

Data Types in Python: Numbers

Complex numbers

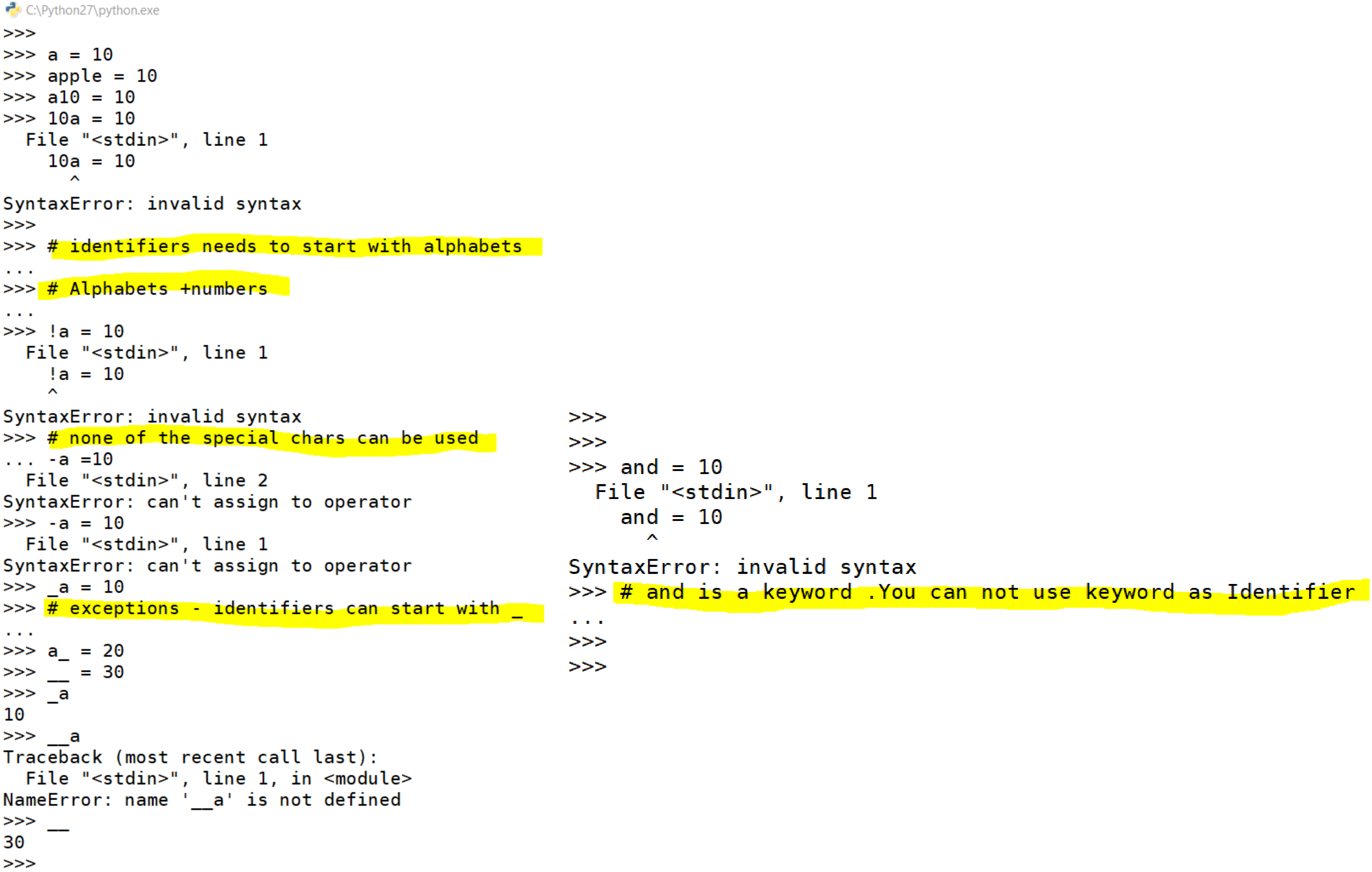
- Are of a form **a + bJ**, where a and b are int or floats and J (or j) represents the square root of -1 (which is an imaginary number). Examples: 3+4j, 3.0+4.0j, 2j
- Must end in j or J
- Complex numbers are not used much in Python programming.

Identifier

- **Python identifiers: Rules for variable names**
  - A python identifier is a name used to identify a variable, function, class, module or other object
  - An identifier starts with a letter **A** to **Z** or **a** to **z** or an underscore **(_)** followed by **zero** or more letters, underscores and digits (0 to 9)
  - Python is a case sensitive language
  - Python does not allow special characters such as @, $ and % within identifiers
- Variables are used by just assigning them a value. No declaration or data type definition is needed/used.

Identifier

- **Identifier naming convention for python**
  - Class names start with an uppercase letter and all other identifiers with lowercase letter
  - Starting an identifier with a single leading underscore indicates by convention that identifier is meant to be private
  - Starting an identifier with two leading underscores indicates a strongly private identifiers
  - If the identifier also ends with two trailing underscores, the identifier is a language – defined special name



C:\Python27\python.exe

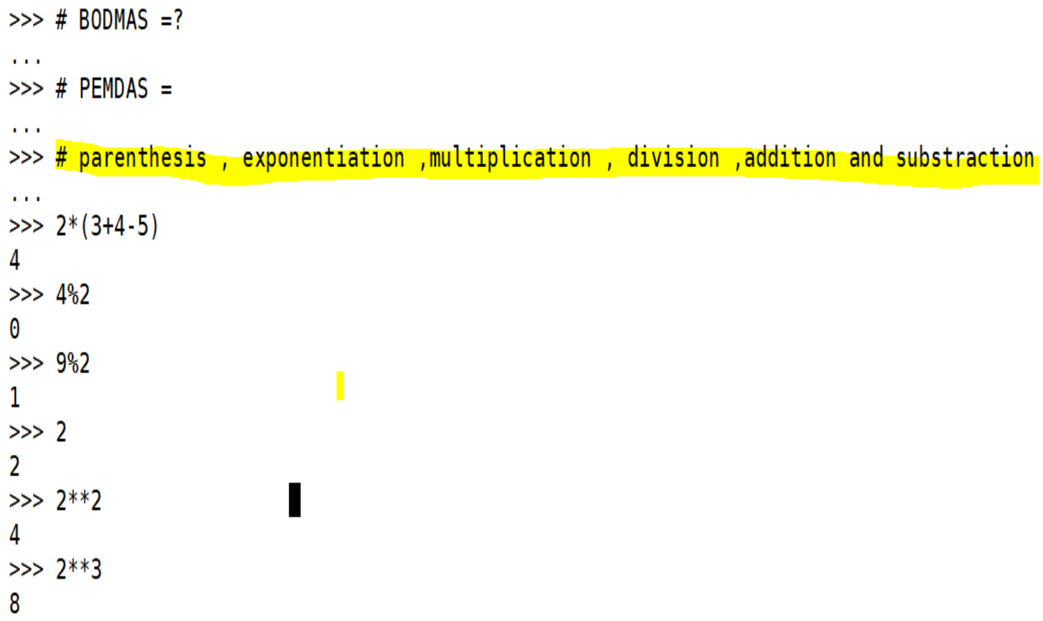
```
>>>
>>> a = 10
>>> apple = 10
>>> a10 = 10
>>> 10a = 10
  File "<stdin>", line 1
    10a = 10
      ^
SyntaxError: invalid syntax
>>>
>>> # identifiers needs to start with alphabets
...
>>> # Alphabets +numbers
...
>>> !a = 10
  File "<stdin>", line 1
    !a = 10
    ^
SyntaxError: invalid syntax
>>> # none of the special chars can be used
... -a =10
  File "<stdin>", line 2
SyntaxError: can't assign to operator
>>> -a = 10
  File "<stdin>", line 1
SyntaxError: can't assign to operator
>>> _a = 10
>>> # exceptions - identifiers can start with _
...
>>> a_ = 20
>>> __ = 30
>>> _a
10
>>> __a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name '__a' is not defined
>>> __
30
>>>
```

```
>>>
>>>
>>> and = 10
  File "<stdin>", line 1
    and = 10
      ^
SyntaxError: invalid syntax
>>> # and is a keyword .You can not use keyword as Identifier
...
>>>
>>>
```

Operations on Numbers

Basic algebraic operations

- Four arithmetic operations : a+b, a-b, a*b, a/b
- Module : a % b
- Exponentiation : a**b
- Other elementary functions are not part of standard Python, but included in packages like NumPy and SciPy



```
>>> # BODMAS =?
...
>>> # PEMDAS =
...
>>> # parenthesis , exponentiation ,multiplication , division ,addition and substraction
...
>>> 2*(3+4-5)
4
>>> 4%2
0
>>> 9%2
1
>>> 2
2
>>> 2**2
4
>>> 2**3
8
```

Comparison operation

- Greater than, less than, etc. a>b, a<b, a<= b, a>= b
- Identity tests: a == b, a!=b

Operations on Numbers

In addition to other Operators:

- Not surprisingly, Python follows the basic *PEMDAS (Parentheses, Exponents, Multiplication, Division, Addition, Subtraction)* order of operation.

- Python supports mixed type math.

Example 1: 100-24*3%5 -> 100-((24*3)%5) -> -> 100-(72%5) -> 100-2 =98

Example 2 : 100-24*(3%5) -> : 100-(24*(3%5)) -> 100 – (24*3) -> 100-72=28

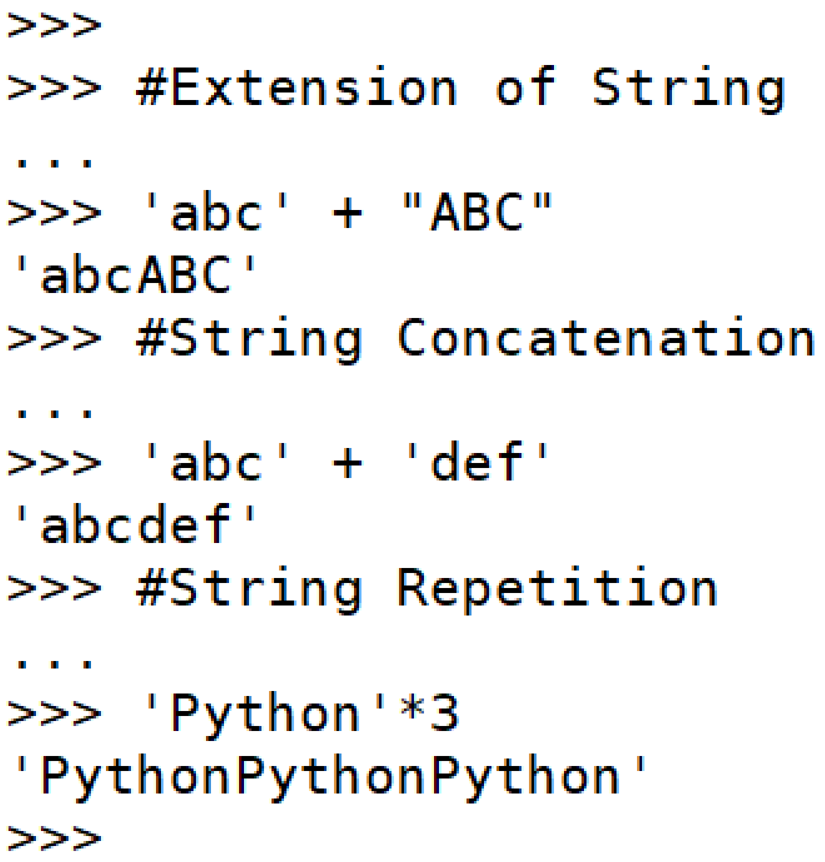
Data Types in Python: Strings

Strings are ordered blocks of text

- Strings are enclosed in single or double quotation marks.
- Double quotation marks allow the user to extend strings over multiple lines without backslashes, which usually signal the continuation of an expression.
- Example: **‘abc’, “ABC”**

Concatenation and repetition

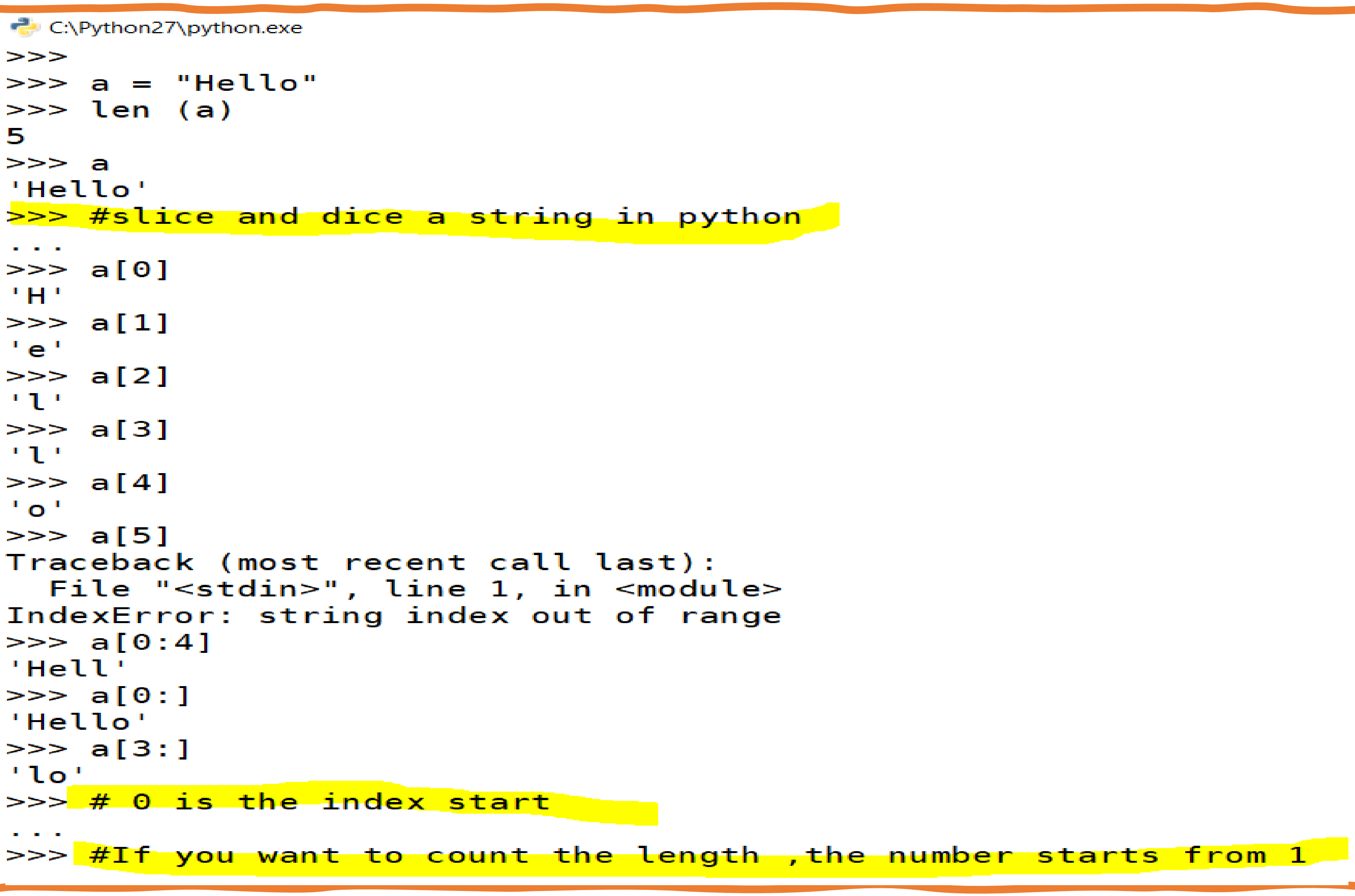
- Strings are concatenated with the **+ sign**
- Strings are repeated with the *** sign**



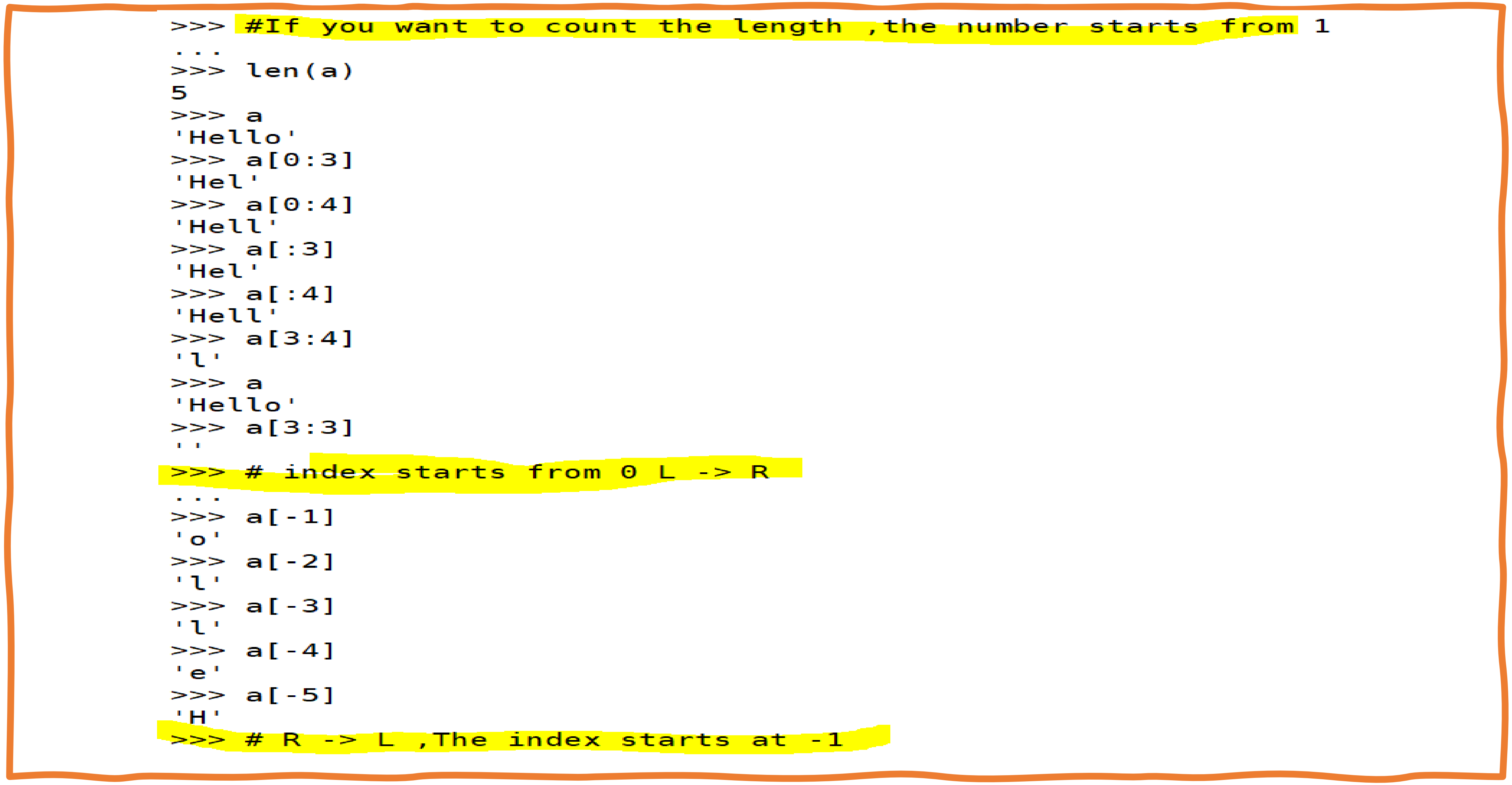
```
>>>
>>> #Extension of String
...
>>> 'abc' + "ABC"
'abcABC'
>>> #String Concatenation
...
>>> 'abc' + 'def'
'abcdef'
>>> #String Repetition
...
>>> 'Python'*3
'PythonPythonPython'
>>>
```

Operations on Strings

- **Indexing and Slicing Operation**
  - Python starts indexing at **0**.
  - A string **s** will have indexes running from **0** to **len(s)-1** (where **len(s)** is the length of **s** in integer quantities.
  - **S[i]** fetches the **ith** element in **s**



```
>>>
>>> a = "Hello"
>>> len (a)
5
>>> a
'Hello'
>>> #slice and dice a string in python
...
>>> a[0]
'H'
>>> a[1]
'e'
>>> a[2]
'l'
>>> a[3]
'l'
>>> a[4]
'o'
>>> a[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> a[0:4]
'Hell'
>>> a[0:]
'Hello'
>>> a[3:]
'lo'
>>> # 0 is the index start
...
>>> #If you want to count the length ,the number starts from 1
```



```
>>> #If you want to count the length ,the number starts from 1
...
>>> len(a)
5
>>> a
'Hello'
>>> a[0:3]
'Hel'
>>> a[0:4]
'Hell'
>>> a[:3]
'Hel'
>>> a[:4]
'Hell'
>>> a[3:4]
'l'
>>> a
'Hello'
>>> a[3:3]
''
>>> # index starts from 0 L -> R
...
>>> a[-1]
'o'
>>> a[-2]
'l'
>>> a[-3]
'l'
>>> a[-4]
'e'
>>> a[-5]
'H'
>>> # R -> L ,The index starts at -1
```

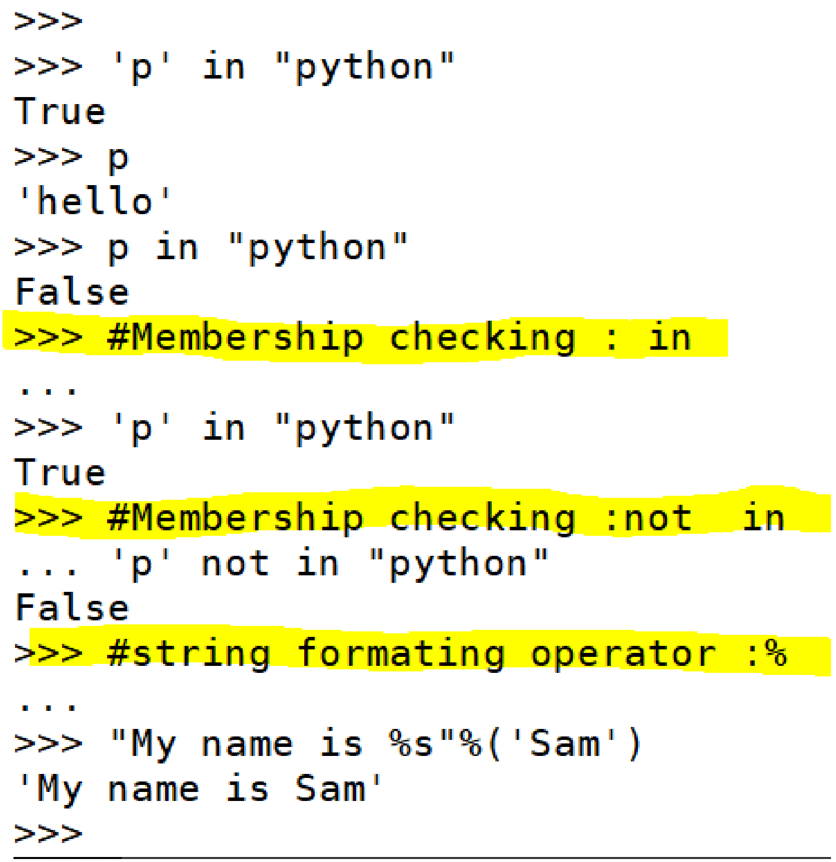
Operations on Strings

Membership Checking

- **In** - Returns true if a character exists in the given string .
- **Not in** – Return true if a character does not exist in the given string

String Formatting Operator : %

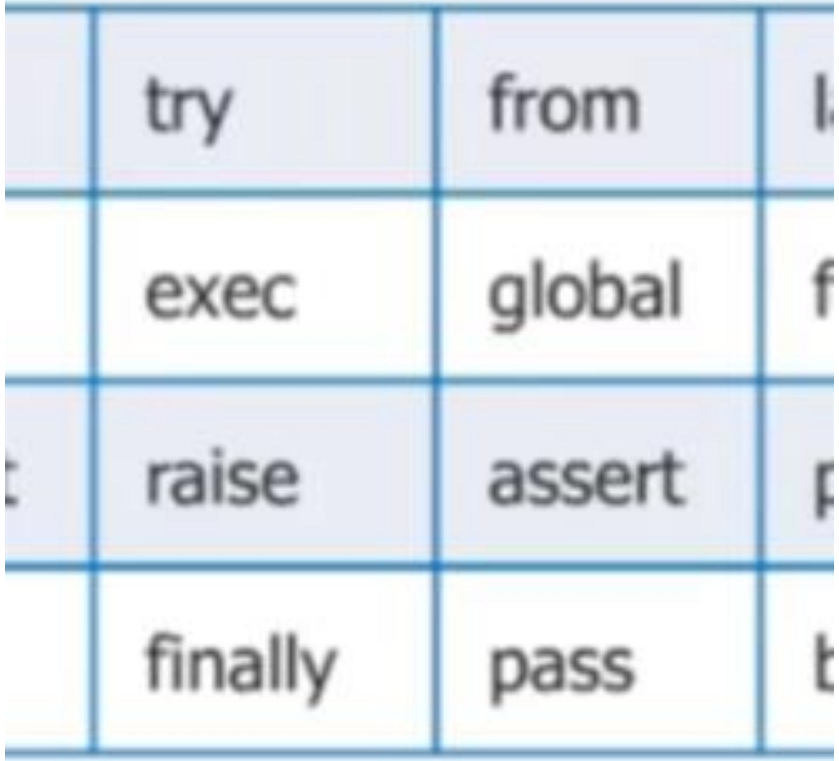
- This operator is unique to strings and Python uses **C-style** formatting to create new, formatted strings. The **%** Operator is used to format a set of variables enclosed in a **“tuple”** (a fixed size list), together with a format string -- **%c, %s, %d** etc.



```
>>>
>>> 'p' in "python"
True
>>> p
'hello'
>>> p in "python"
False
>>> #Membership checking : in
...
>>> 'p' in "python"
True
>>> #Membership checking :not  in
... 'p' not in "python"
False
>>> #string formating operator :%
...
>>> "My name is %s"%('Sam')
'My name is Sam'
>>>
```

Reserved Keywords

- Reserved keywords are the reserved words in python which can not be used as :
  - Variable name
  - Function name or
  - Any other identifier
- They are used to define the syntax and structure of the python language
- All the python keywords contain lowercase letter only.

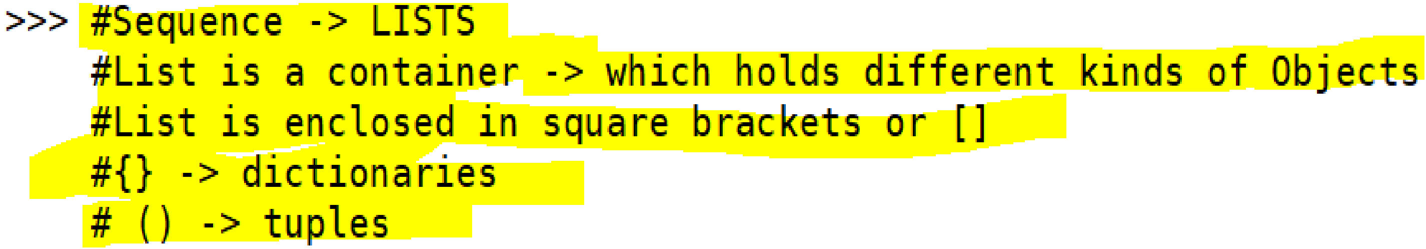


try	from
exec	global
raise	assert
finally	pass

Data Types in Python: List

- The list type is a container which holds a number of other objects, in a given order.
- The list type implements the sequence protocol, and it also allow you to add and remove objects from the sequence.
- A list is an ordered set of elements enclosed in square brackets.
- Simple definition of list -> **li = []**

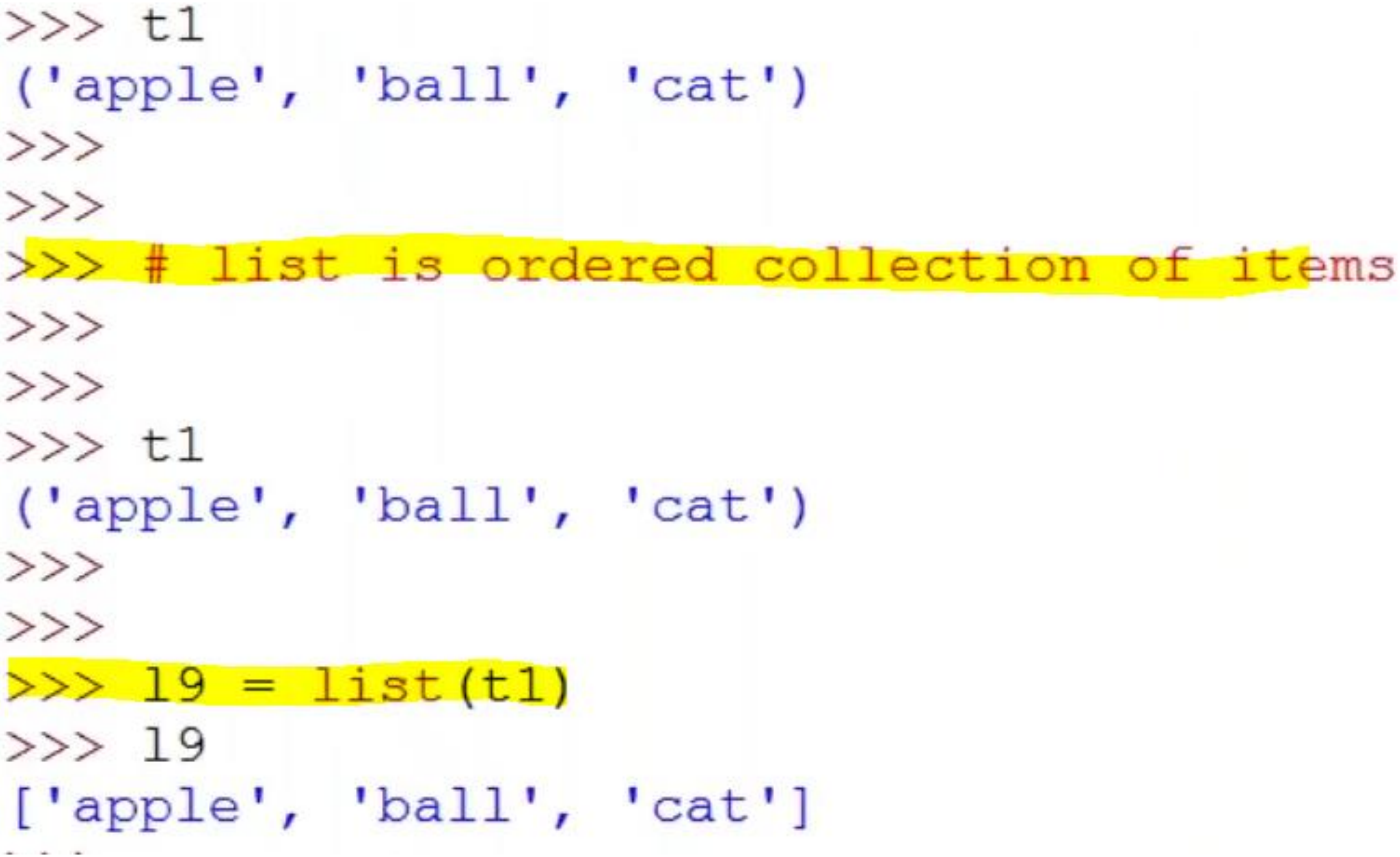
Using built in LIST type object:



```
>>> #Sequence -> LISTS
    #List is a container -> which holds different kinds of Objects
    #List is enclosed in square brackets or []
    #{} -> dictionaries
    # () -> tuples
```

Data Types in Python: List - Access

- Accessing elements in a list:
  - n=len(li)
  - Item = li[index] **#Indexing**
  - Slice = li [start:stop] **#Slicing**



```
>>> t1
('apple', 'ball', 'cat')
>>>
>>>
>>> # list is ordered collection of items
>>>
>>>
>>> t1
('apple', 'ball', 'cat')
>>>
>>>
>>> l9 = list(t1)
>>> l9
['apple', 'ball', 'cat']
```

Data Types in Python: List - Indexing

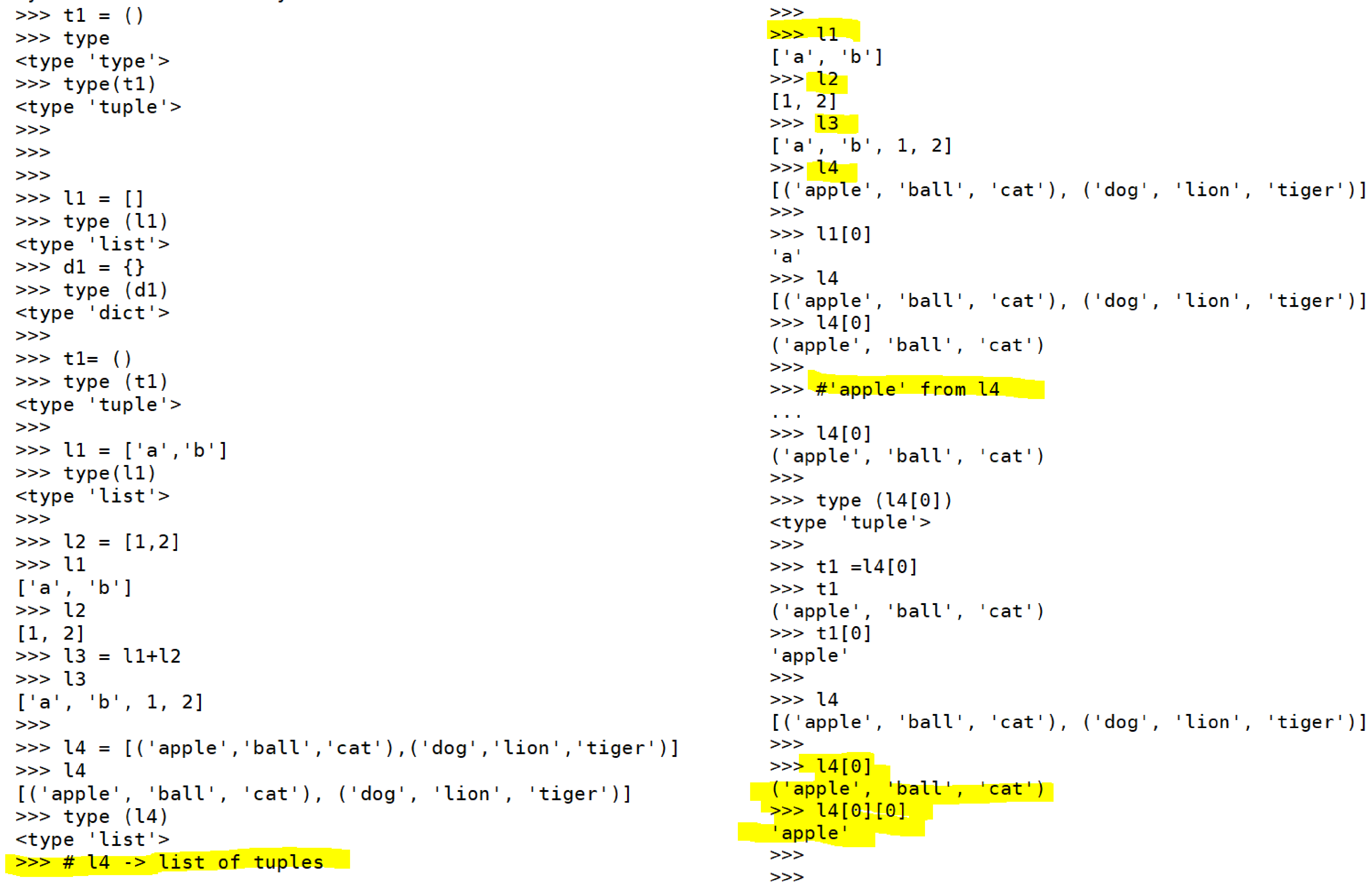
- **List[i]** returns the value at index I. Where I is an integer

- A negative index accesses elements from the end of the list counting backwards. The last element of any nonempty list is always **list[-1]**

- Python raises an **IndexError** exception, if the index is outside the list

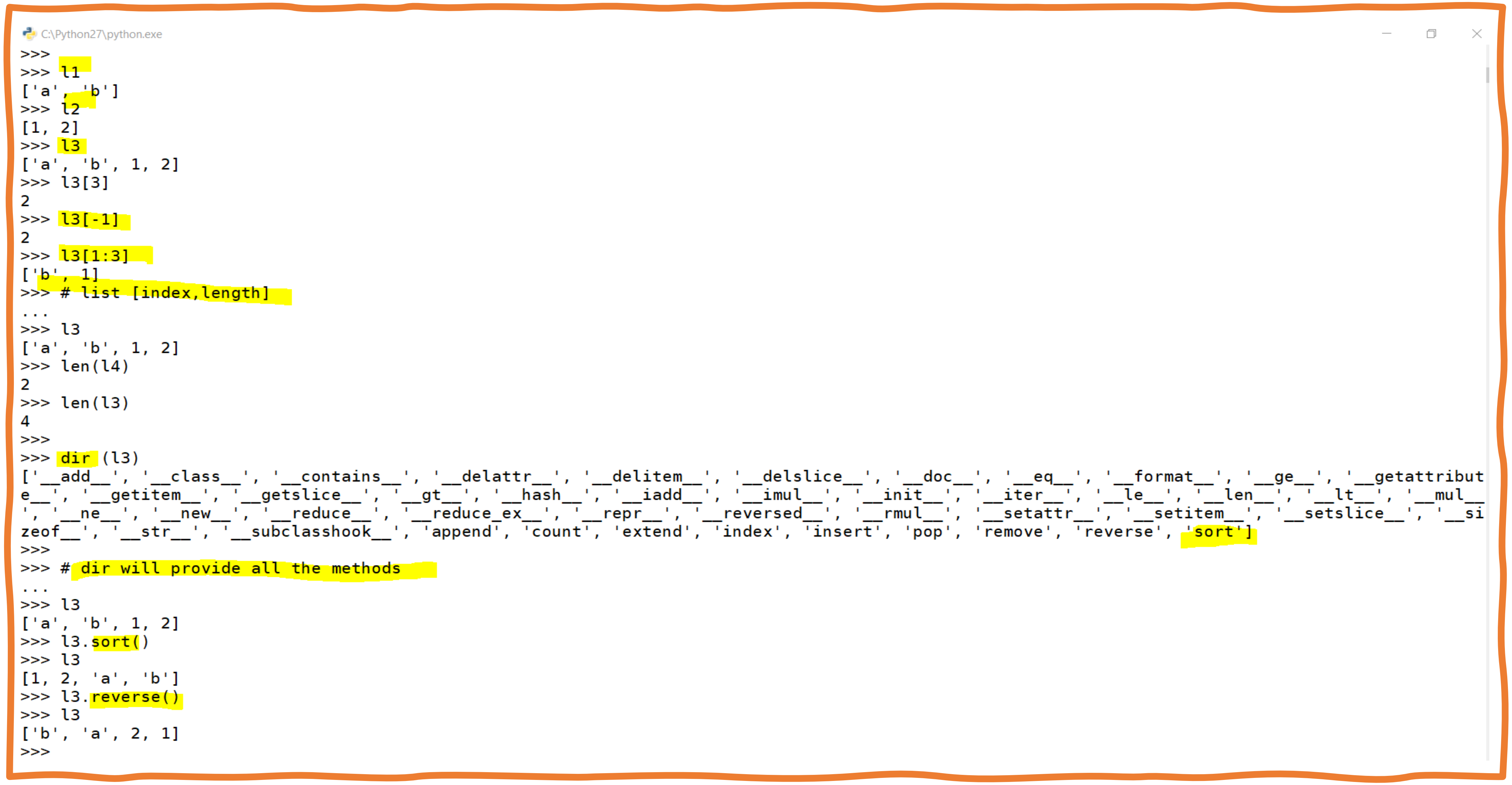
Data Types in Python: List - Slicing

- A subset of list is called **“slice”**
- You can get a subset of list, called a **“slice”**, by specifying two indices
- Slicing works if one or both of the slice indices is negative

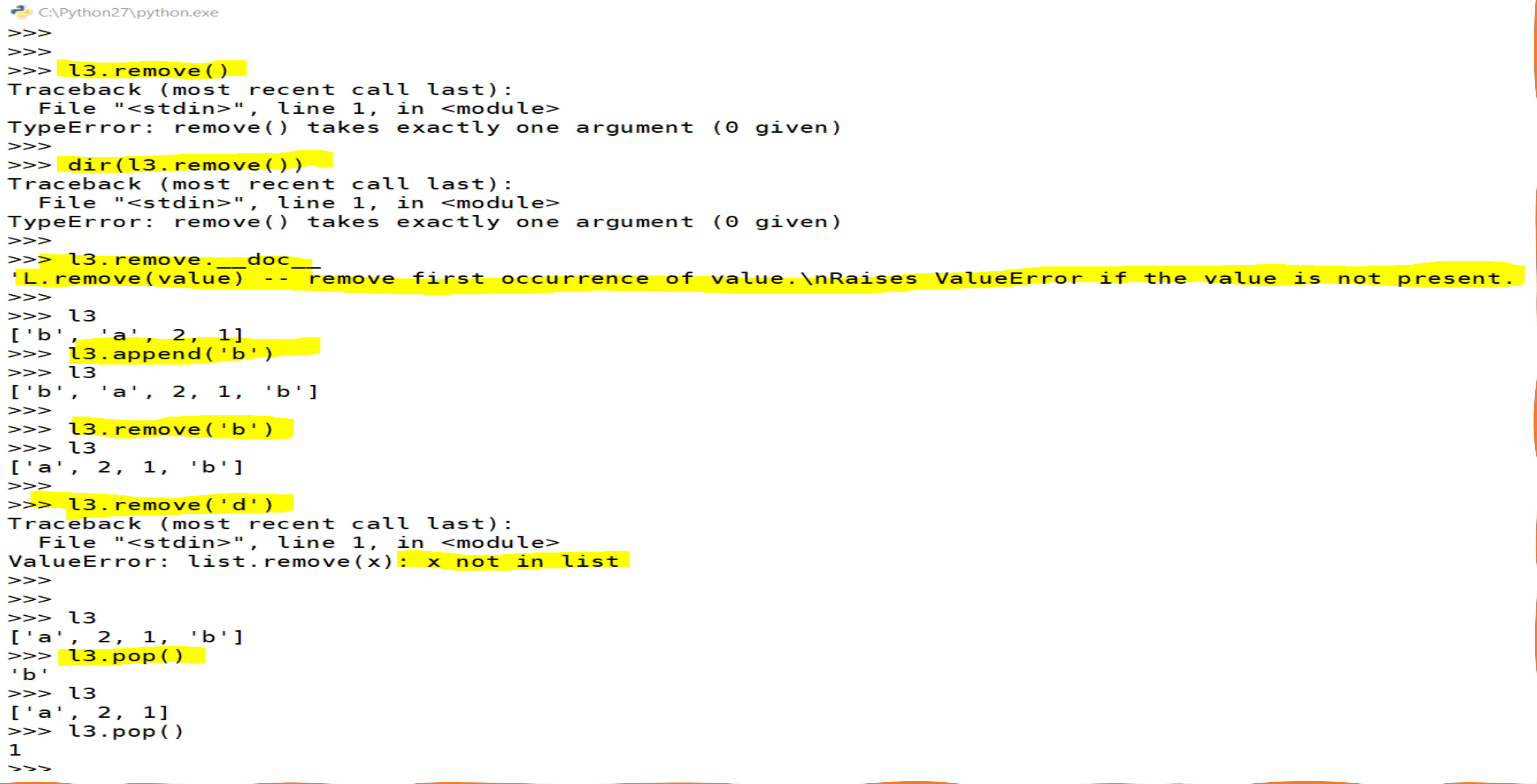


```
>>> t1 = ()
>>> type
<type 'type'>
>>> type(t1)
<type 'tuple'>
>>>
>>>
>>>
>>> l1 = []
>>> type (l1)
<type 'list'>
>>> d1 = {}
>>> type (d1)
<type 'dict'>
>>>
>>> t1= ()
>>> type (t1)
<type 'tuple'>
>>>
>>> l1 = ['a','b']
>>> type(l1)
<type 'list'>
>>>
>>> l2 = [1,2]
>>> l1
['a', 'b']
>>> l2
[1, 2]
>>> l3 = l1+l2
>>> l3
['a', 'b', 1, 2]
>>>
>>> l4 = [('apple','ball','cat'),('dog','lion','tiger')]
>>> l4
[('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>> type (l4)
<type 'list'>
>>> # l4 -> list of tuples
```

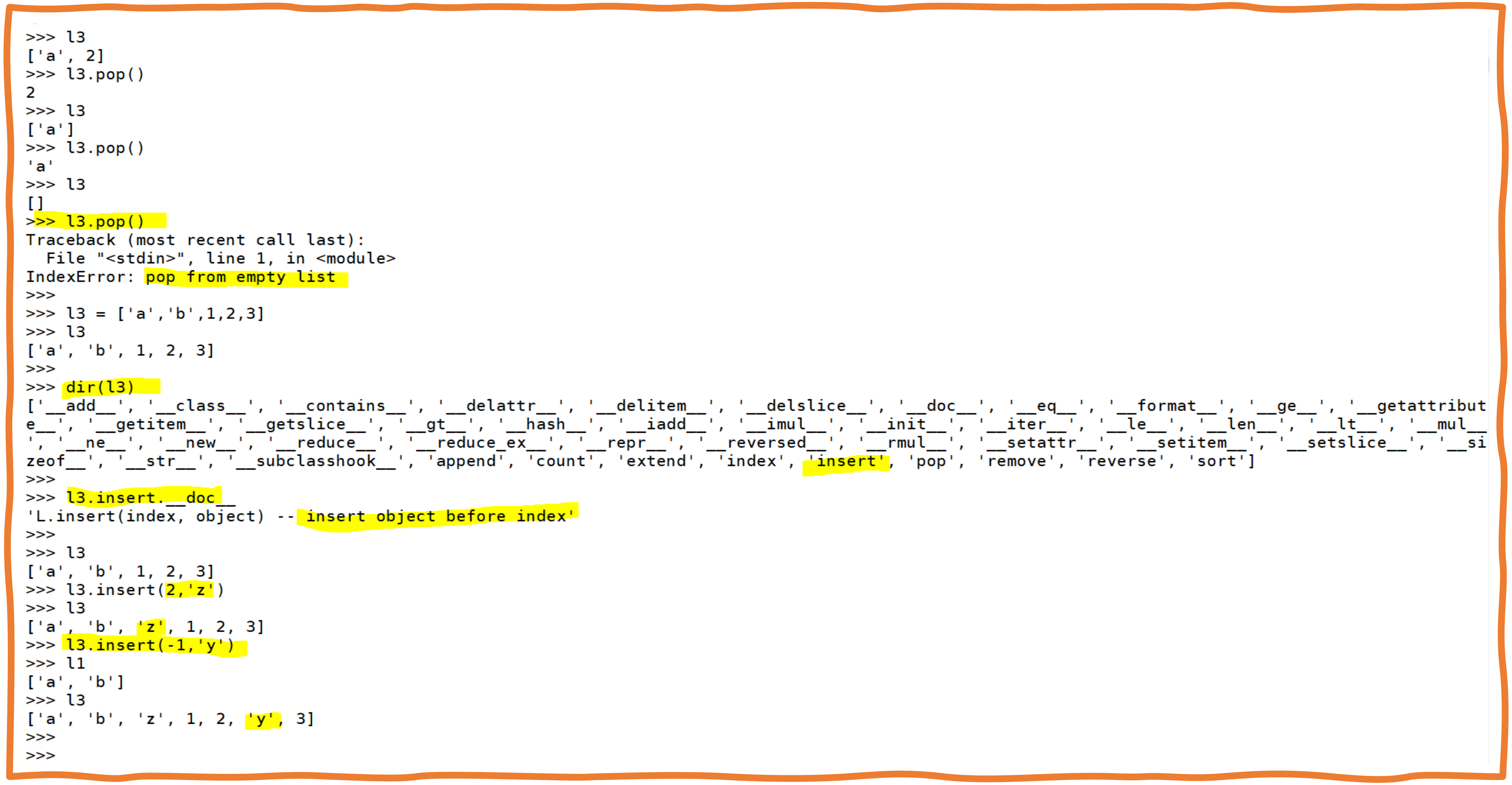
```
>>>
>>> l1
['a', 'b']
>>> l2
[1, 2]
>>> l3
['a', 'b', 1, 2]
>>> l4
[('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>>
>>> l1[0]
'a'
>>> l4
[('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>> l4[0]
('apple', 'ball', 'cat')
>>>
>>> #'apple' from l4
...
>>> l4[0]
('apple', 'ball', 'cat')
>>>
>>> type (l4[0])
<type 'tuple'>
>>>
>>> t1 =l4[0]
>>> t1
('apple', 'ball', 'cat')
>>> t1[0]
'apple'
>>>
>>> l4
[('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>>
>>> l4[0]
('apple', 'ball', 'cat')
>>> l4[0][0]
'apple'
>>>
>>>
```



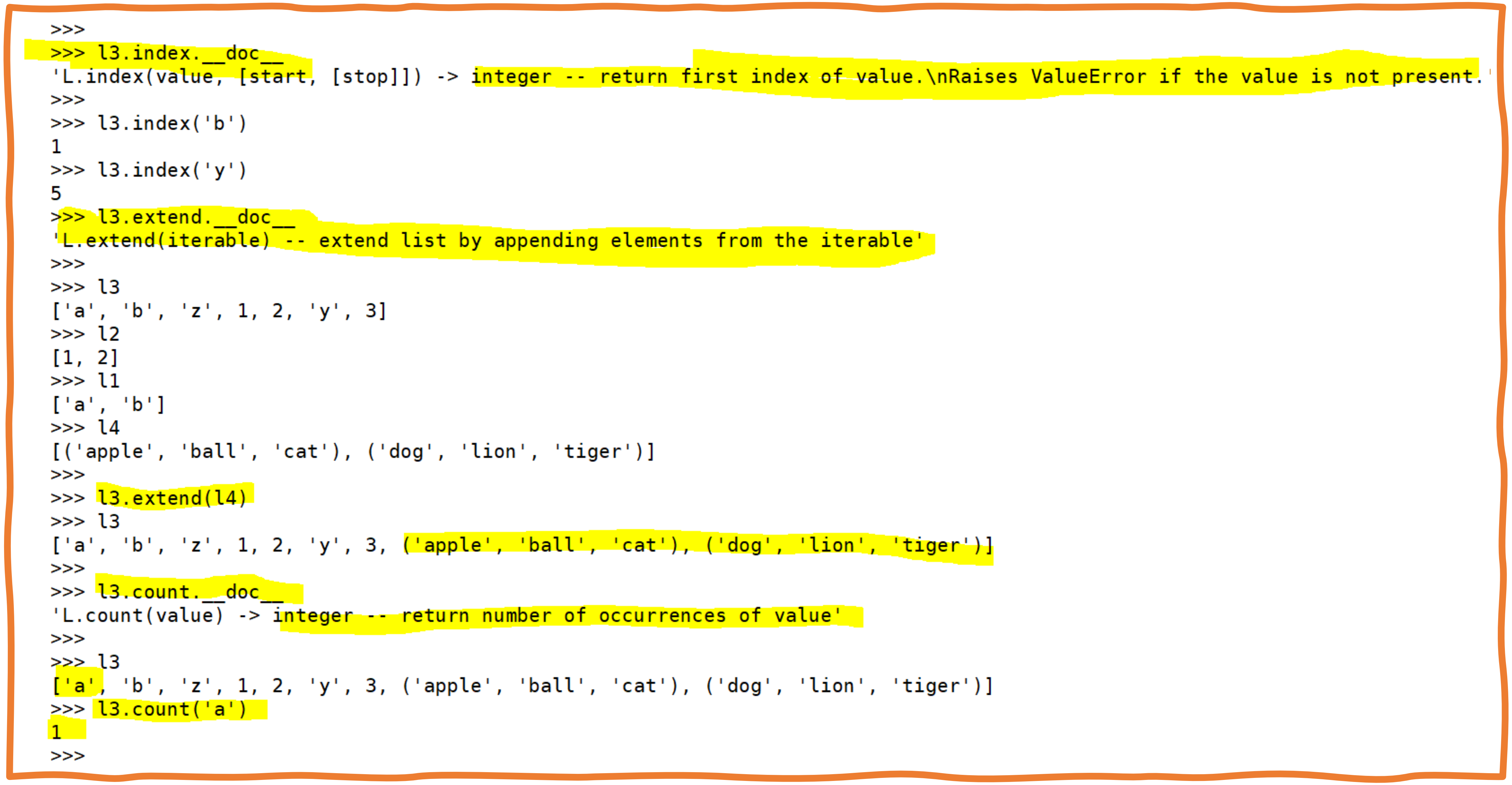
```
>>>
>>> l1
['a', 'b']
>>> l2
[1, 2]
>>> l3
['a', 'b', 1, 2]
>>> l3[3]
2
>>> l3[-1]
2
>>> l3[1:3]
['b', 1]
>>> # list [index,length]
...
>>> l3
['a', 'b', 1, 2]
>>> len(l4)
2
>>> len(l3)
4
>>>
>>> dir (l3)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__delslice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribut
e__', '__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__
', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__setslice__', '__si
zeof__', '__str__', '__subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
>>> # dir will provide all the methods
...
>>> l3
['a', 'b', 1, 2]
>>> l3.sort()
>>> l3
[1, 2, 'a', 'b']
>>> l3.reverse()
>>> l3
['b', 'a', 2, 1]
>>>
```



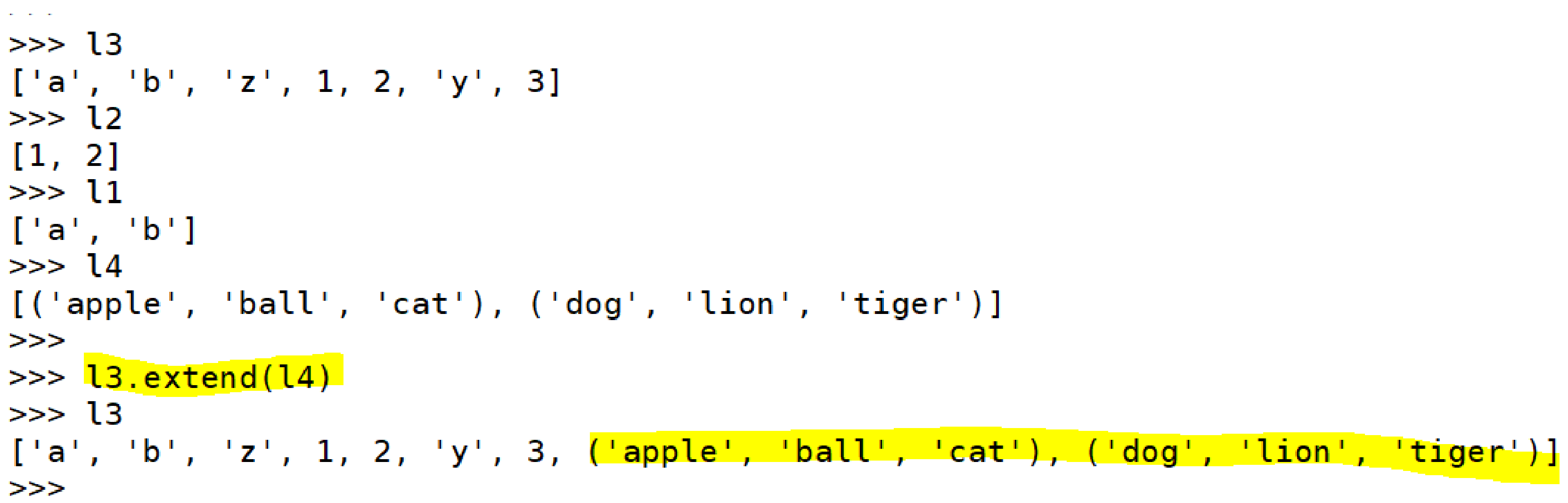
```
>>>
>>>
>>> l3.remove()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: remove() takes exactly one argument (0 given)
>>>
>>> dir(l3.remove())
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: remove() takes exactly one argument (0 given)
>>>
>>> l3.remove.__doc__
'L.remove(value) -- remove first occurrence of value.\nRaises ValueError if the value is not present.
>>>
>>> l3
['b', 'a', 2, 1]
>>> l3.append('b')
>>> l3
['b', 'a', 2, 1, 'b']
>>>
>>> l3.remove('b')
>>> l3
['a', 2, 1, 'b']
>>>
>>> l3.remove('d')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
>>>
>>>
>>> l3
['a', 2, 1, 'b']
>>> l3.pop()
'b'
>>> l3
['a', 2, 1]
>>> l3.pop()
1
>>>
```



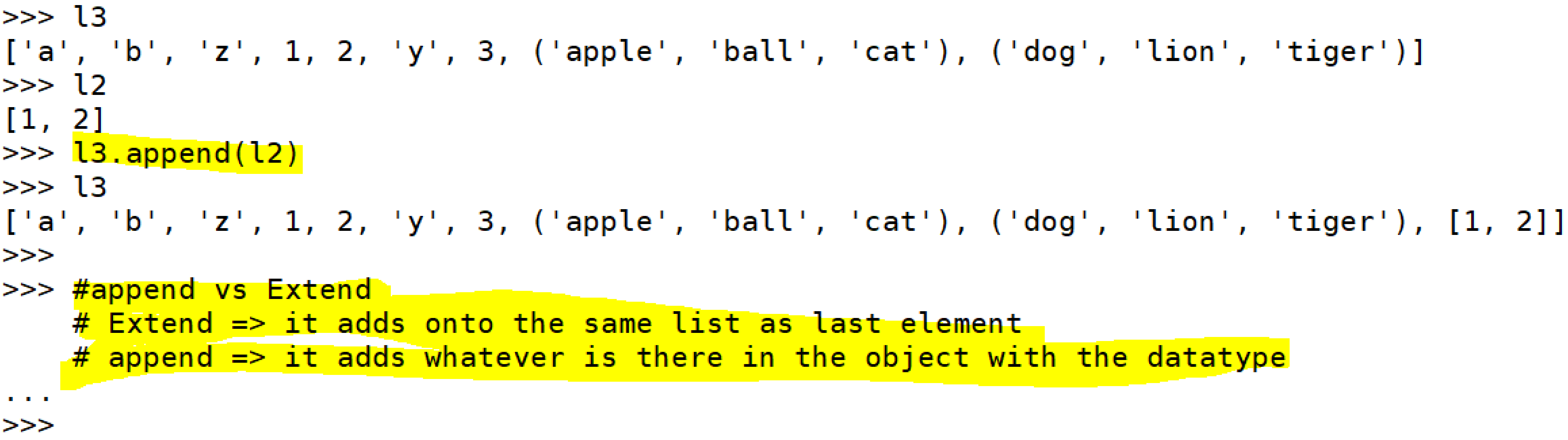
```
>>> l3
['a', 2]
>>> l3.pop()
2
>>> l3
['a']
>>> l3.pop()
'a'
>>> l3
[]
>>> l3.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: pop from empty list
>>>
>>> l3 = ['a','b',1,2,3]
>>> l3
['a', 'b', 1, 2, 3]
>>>
>>> dir(l3)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__delslice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribut
e__', '__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__
', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__setslice__', '__si
zeof__', '__str__', '__subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
>>> l3.insert.__doc__
'L.insert(index, object) -- insert object before index'
>>>
>>> l3
['a', 'b', 1, 2, 3]
>>> l3.insert(2,'z')
>>> l3
['a', 'b', 'z', 1, 2, 3]
>>> l3.insert(-1,'y')
>>> l1
['a', 'b']
>>> l3
['a', 'b', 'z', 1, 2, 'y', 3]
>>>
>>>
```



```
>>>
>>> l3.index.__doc__
'L.index(value, [start, [stop]]) -> integer -- return first index of value.\nRaises ValueError if the value is not present.'
>>>
>>> l3.index('b')
1
>>> l3.index('y')
5
>>> l3.extend.__doc__
'L.extend(iterable) -- extend list by appending elements from the iterable'
>>>
>>> l3
['a', 'b', 'z', 1, 2, 'y', 3]
>>> l2
[1, 2]
>>> l1
['a', 'b']
>>> l4
[('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>>
>>> l3.extend(l4)
>>> l3
['a', 'b', 'z', 1, 2, 'y', 3, ('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>>
>>> l3.count.__doc__
'L.count(value) -> integer -- return number of occurrences of value'
>>>
>>> l3
['a', 'b', 'z', 1, 2, 'y', 3, ('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>> l3.count('a')
1
>>>
```



```
...
>>> l3
['a', 'b', 'z', 1, 2, 'y', 3]
>>> l2
[1, 2]
>>> l1
['a', 'b']
>>> l4
[('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>>
>>> l3.extend(l4)
>>> l3
['a', 'b', 'z', 1, 2, 'y', 3, ('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>>
```



```
>>> l3
['a', 'b', 'z', 1, 2, 'y', 3, ('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger')]
>>> l2
[1, 2]
>>> l3.append(l2)
>>> l3
['a', 'b', 'z', 1, 2, 'y', 3, ('apple', 'ball', 'cat'), ('dog', 'lion', 'tiger'), [1, 2]]
>>>
>>> #append vs Extend
    # Extend => it adds onto the same list as last element
    # append => it adds whatever is there in the object with the datatype
...
>>>
```

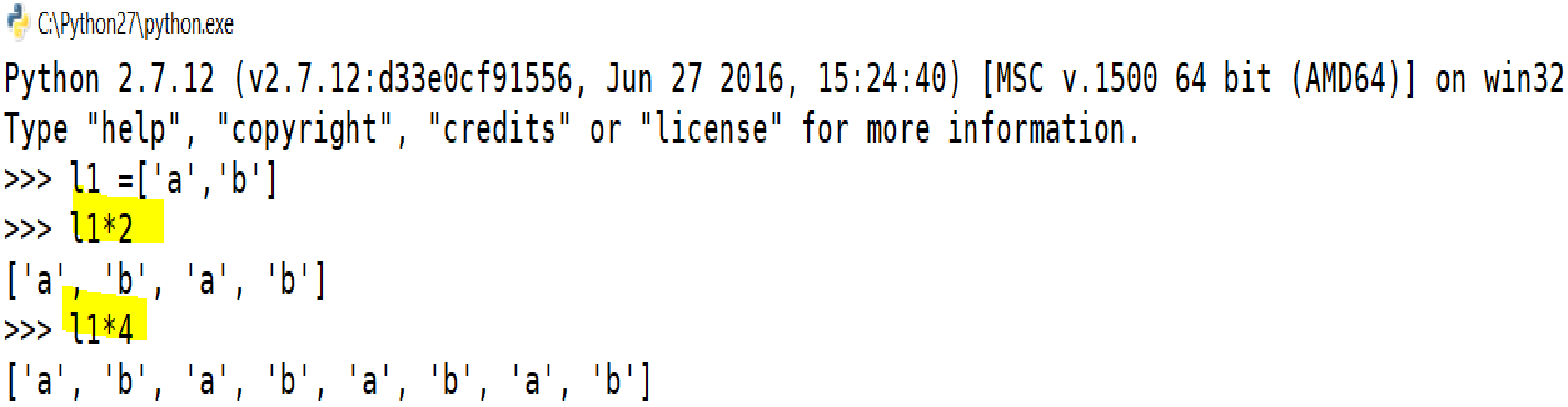
Extend Home: break the house

Append Home: do not break but just add to it.

Data Types in Python: List - Operator

- Lists can also be concatenated with the **+ operator.**

- **list = list + otherlist** has the same result as **list.extend(otherlist)**



C:\Python27\python.exe

```
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:24:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> l1 =['a','b']
>>> l1*2
['a', 'b', 'a', 'b']
>>> l1*4
['a', 'b', 'a', 'b', 'a', 'b', 'a', 'b']
```

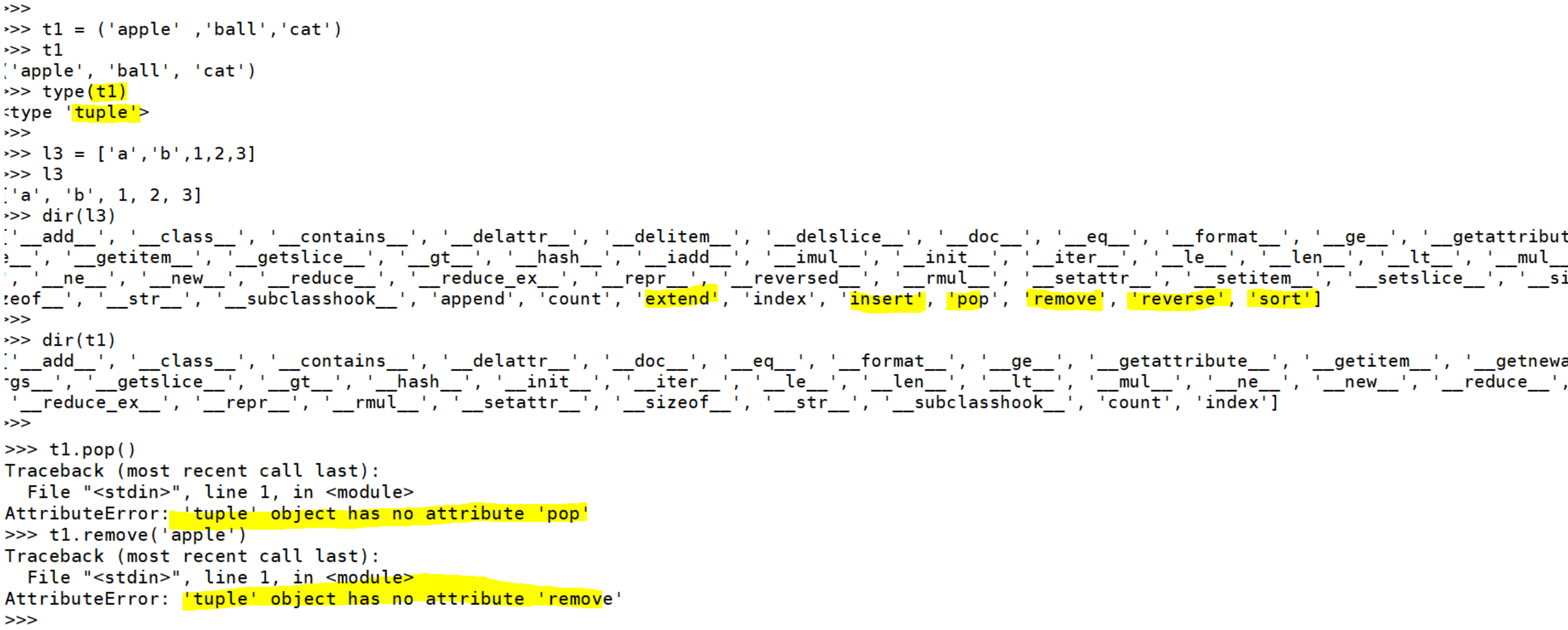
Data Types in Python: Tuple

- A Tuple is an **immutable list**. A tuple can not be changed in any way once it is created.

- A Tuples is defined in the same way as a list except that the whole set of elements are enclosed in parentheses instead of square brackets.

Compare t1(Tuple) and l3(list)

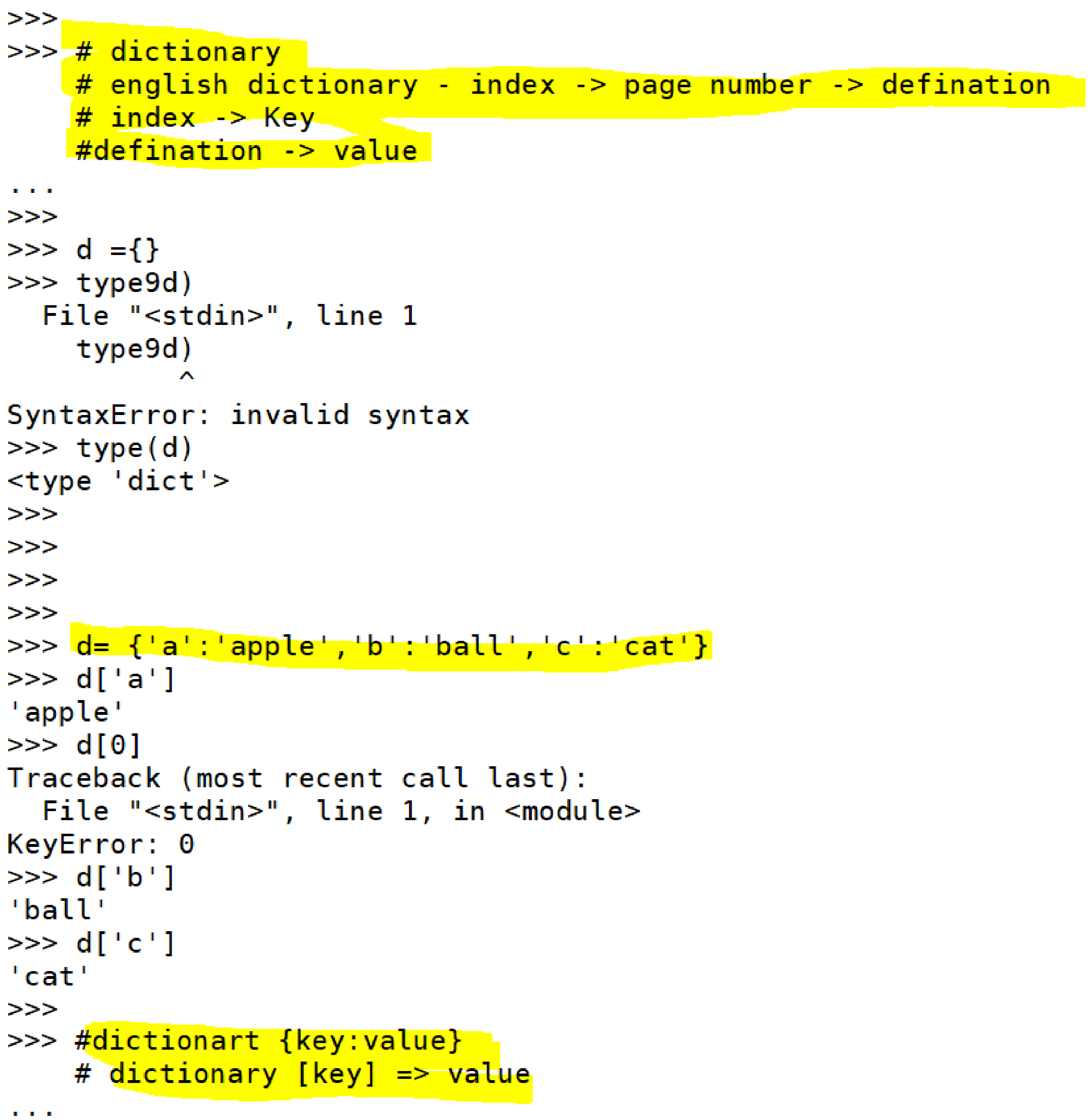
- As you can see you can not edit t1(extend, insert, pop, remove, reverse, sort)



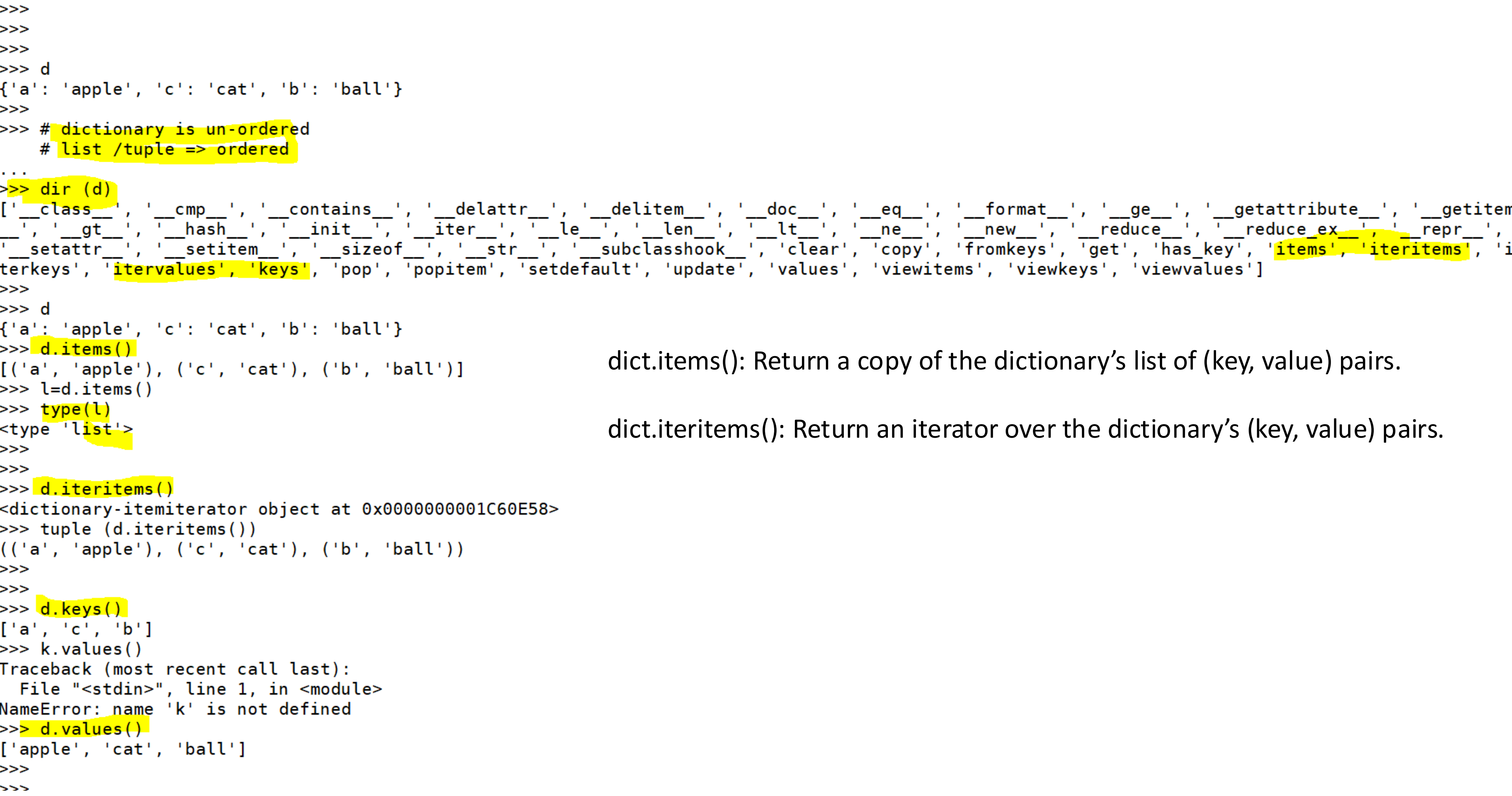
```
>>>
>>> t1 = ('apple' ,'ball','cat')
>>> t1
('apple', 'ball', 'cat')
>>> type(t1)
<type 'tuple'>
>>>
>>> l3 = ['a','b',1,2,3]
>>> l3
['a', 'b', 1, 2, 3]
>>> dir(l3)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__delslice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribut
e__', '__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul_
', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__setslice__', '__si
zeof__', '__str__', '__subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
>>> dir(t1)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewa
gs__', '__getslice__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
>>>

>>> t1.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'pop'
>>> t1.remove('apple')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'remove'
>>>
```

Data Types in Python: Dictionaries

- Collection of arbitrary objects which is unordered, changeable and indexed
- Written in curly brackets, and have keys and values
- Variable-length, heterogenous, and arbitrary nestable
- Mutable mapping
- Table of object references (hash tables)



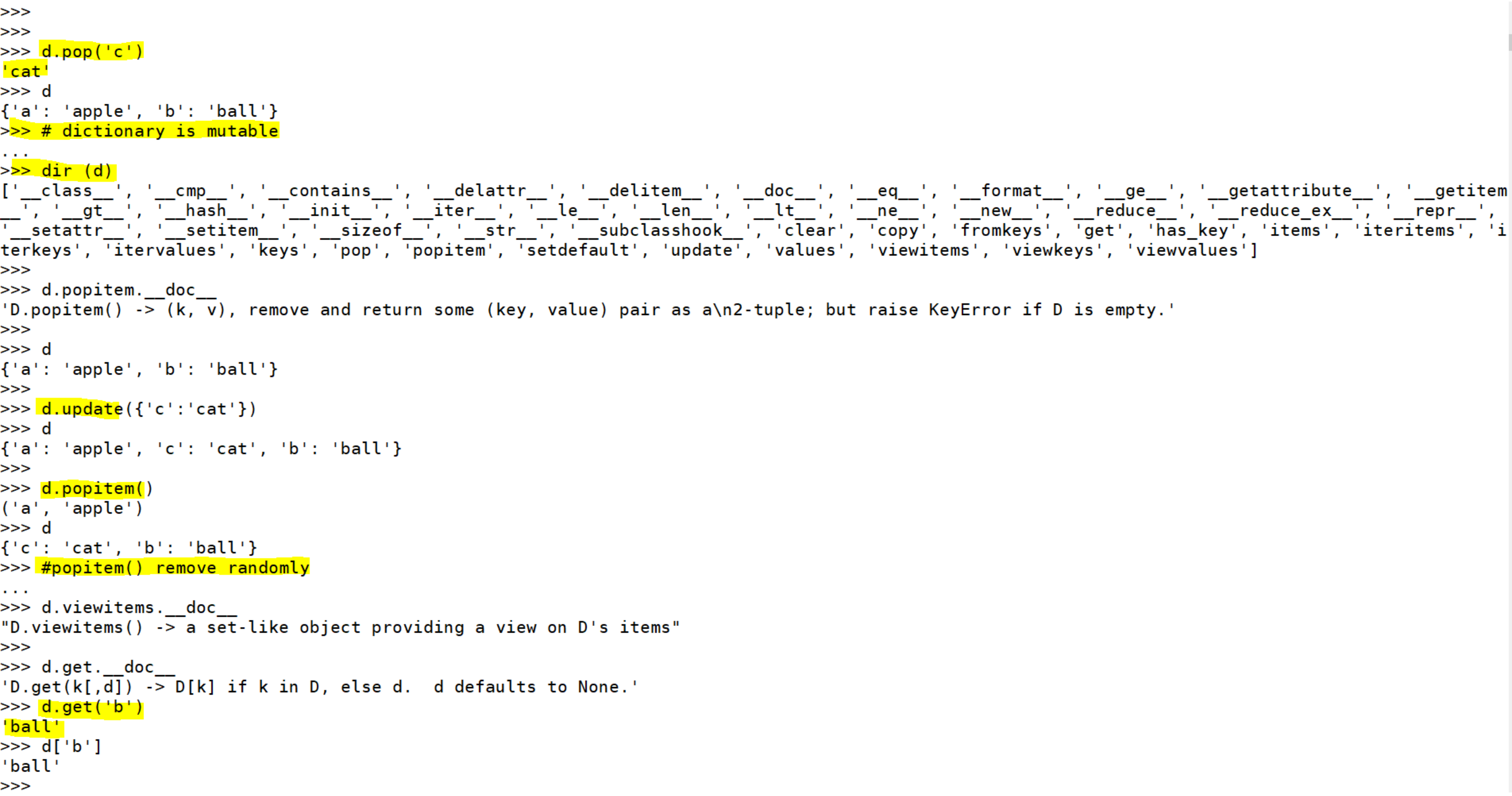
```
>>>
>>> # dictionary
    # english dictionary - index -> page number -> defination
    # index -> Key
    #defination -> value
...
>>>
>>> d ={}
>>> type9d)
  File "<stdin>", line 1
    type9d)
          ^
SyntaxError: invalid syntax
>>> type(d)
<type 'dict'>
>>>
>>>
>>>
>>>
>>> d= {'a':'apple','b':'ball','c':'cat'}
>>> d['a']
'apple'
>>> d[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
>>> d['b']
'ball'
>>> d['c']
'cat'
>>>
>>> #dictionart {key:value}
    # dictionary [key] => value
...
```



```
>>>
>>>
>>>
>>> d
{'a': 'apple', 'c': 'cat', 'b': 'ball'}
>>>
>>> # dictionary is un-ordered
    # list /tuple => ordered
...
>>> dir (d)
['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem
__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'has_key', 'items', 'iteritems', 'i
terkeys', 'itervalues', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values', 'viewitems', 'viewkeys', 'viewvalues']
>>>
>>> d
{'a': 'apple', 'c': 'cat', 'b': 'ball'}
>>> d.items()
[('a', 'apple'), ('c', 'cat'), ('b', 'ball')]
>>> l=d.items()
>>> type(l)
<type 'list'>
>>>
>>>
>>> d.iteritems()
<dictionary-itemiterator object at 0x0000000001C60E58>
>>> tuple (d.iteritems())
(('a', 'apple'), ('c', 'cat'), ('b', 'ball'))
>>>
>>>
>>> d.keys()
['a', 'c', 'b']
>>> k.values()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'k' is not defined
>>> d.values()
['apple', 'cat', 'ball']
>>>
>>>
```

dict.items(): Return a copy of the dictionary’s list of (key, value) pairs.

dict.iteritems(): Return an iterator over the dictionary’s (key, value) pairs.



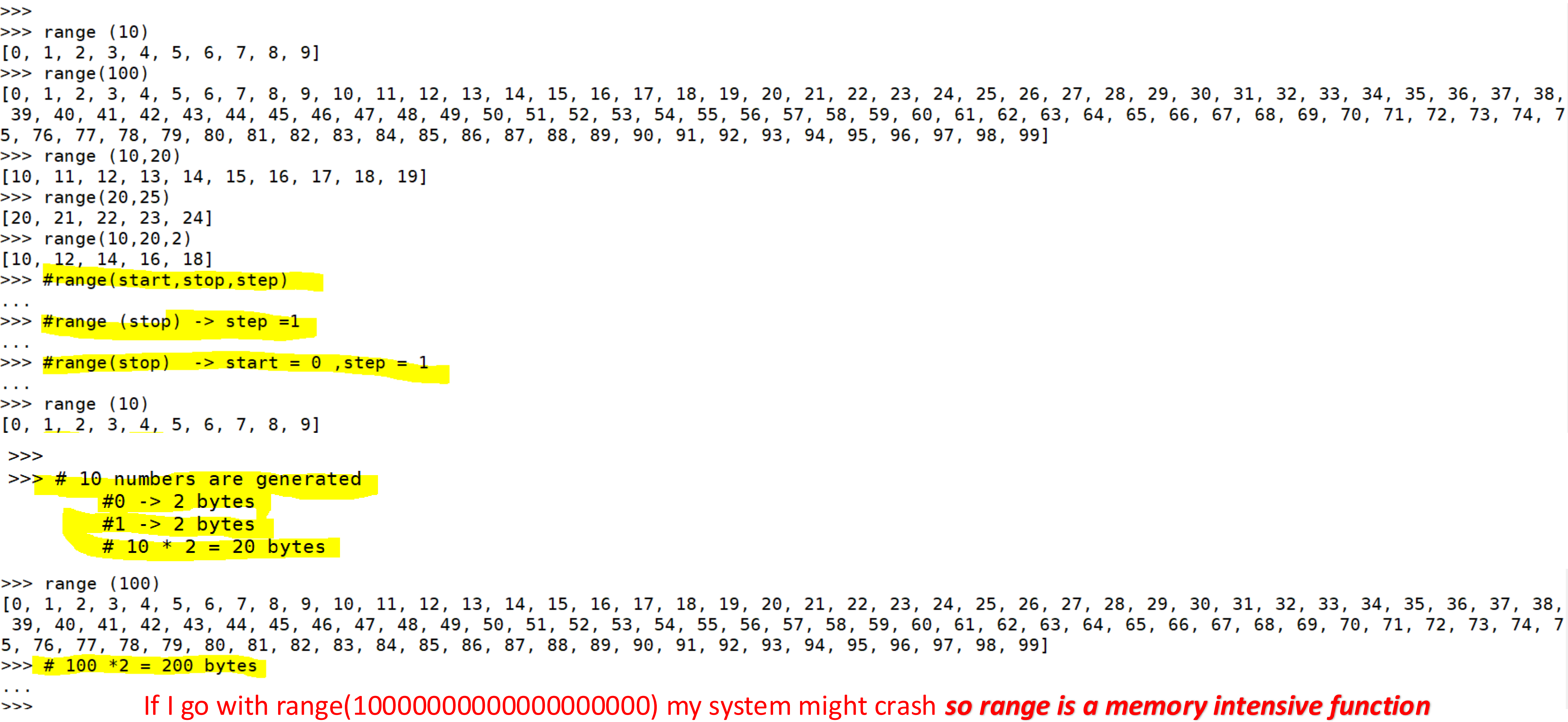
```
>>>
>>>
>>> d.pop('c')
'cat'
>>> d
{'a': 'apple', 'b': 'ball'}
>>> # dictionary is mutable
...
>>> dir (d)
['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem
__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'has_key', 'items', 'iteritems', 'i
terkeys', 'itervalues', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values', 'viewitems', 'viewkeys', 'viewvalues']
>>>
>>> d.popitem.__doc__
'D.popitem() -> (k, v), remove and return some (key, value) pair as a\n2-tuple; but raise KeyError if D is empty.'
>>>
>>> d
{'a': 'apple', 'b': 'ball'}
>>>
>>> d.update({'c':'cat'})
>>> d
{'a': 'apple', 'c': 'cat', 'b': 'ball'}
>>>
>>> d.popitem()
('a', 'apple')
>>> d
{'c': 'cat', 'b': 'ball'}
>>> #popitem() remove randomly
...
>>> d.viewitems.__doc__
"D.viewitems() -> a set-like object providing a view on D's items"
>>>
>>> d.get.__doc__
'D.get(k[,d]) -> D[k] if k in D, else d.  d defaults to None.'
>>> d.get('b')
'ball'
>>> d['b']
'ball'
>>>
```

Range Function

- **range()** generates lists containing arithmetic progression

- 3 variations of **range()** function:
  - **range(stop)** – starts from 0 till (stop 1)
  - **range(start, stop)** – end at (stop 1)
  - **range(start, stop, step)** - Step can not be 0, default is 1

Range

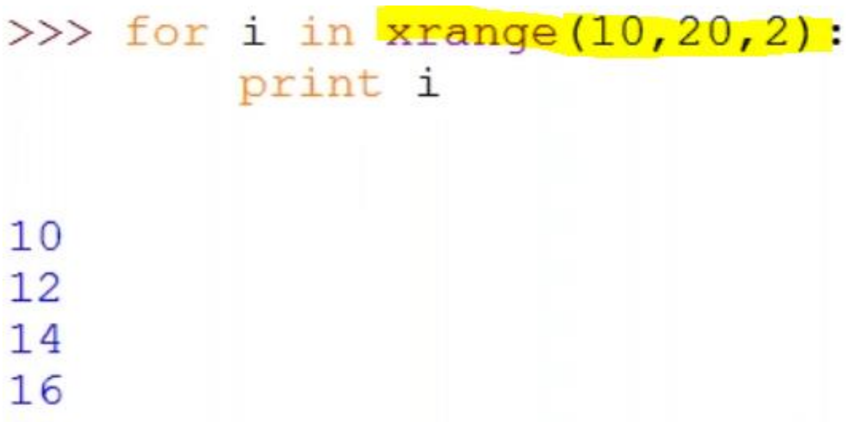


```
>>>
>>> range (10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(100)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 7
5, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
>>> range (10,20)
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> range(20,25)
[20, 21, 22, 23, 24]
>>> range(10,20,2)
[10, 12, 14, 16, 18]
>>> #range(start,stop,step)
...
>>> #range (stop) -> step =1
...
>>> #range(stop)  -> start = 0 ,step = 1
...
>>> range (10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

 >>>
 >>> # 10 numbers are generated
         #0 -> 2 bytes
         #1 -> 2 bytes
         # 10 * 2 = 20 bytes

>>> range (100)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 7
5, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
>>> # 100 *2 = 200 bytes
...
>>>
```

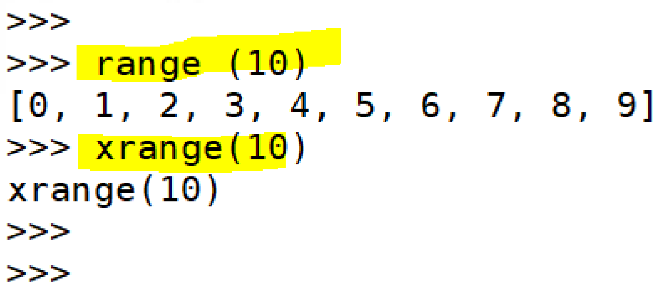
If I go with range(1000000000000000000) my system might crash ***so range is a memory intensive function***

range vs xrange (xrange not in Python 3)

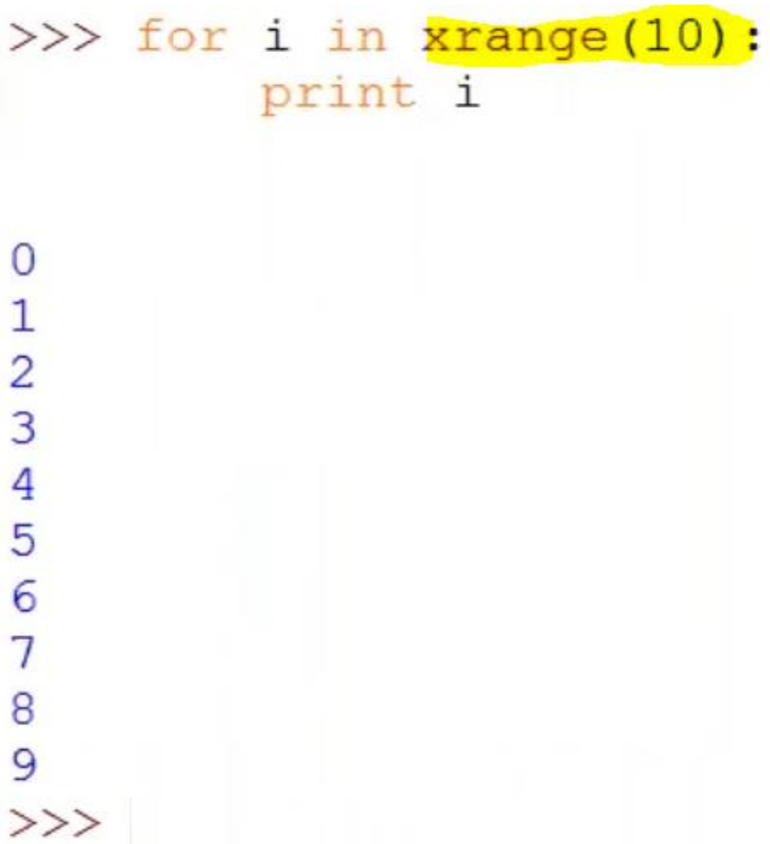
- ***range is a memory intensive function***
- ***Range returns a list however xrange returns an object***
- ***xrange takes only 2 bytes***
- 100000 numbers -> 2 bytes in xrange



```
>>> for i in xrange(10,20,2):
        print i



10
12
14
16
18
>>>
```



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



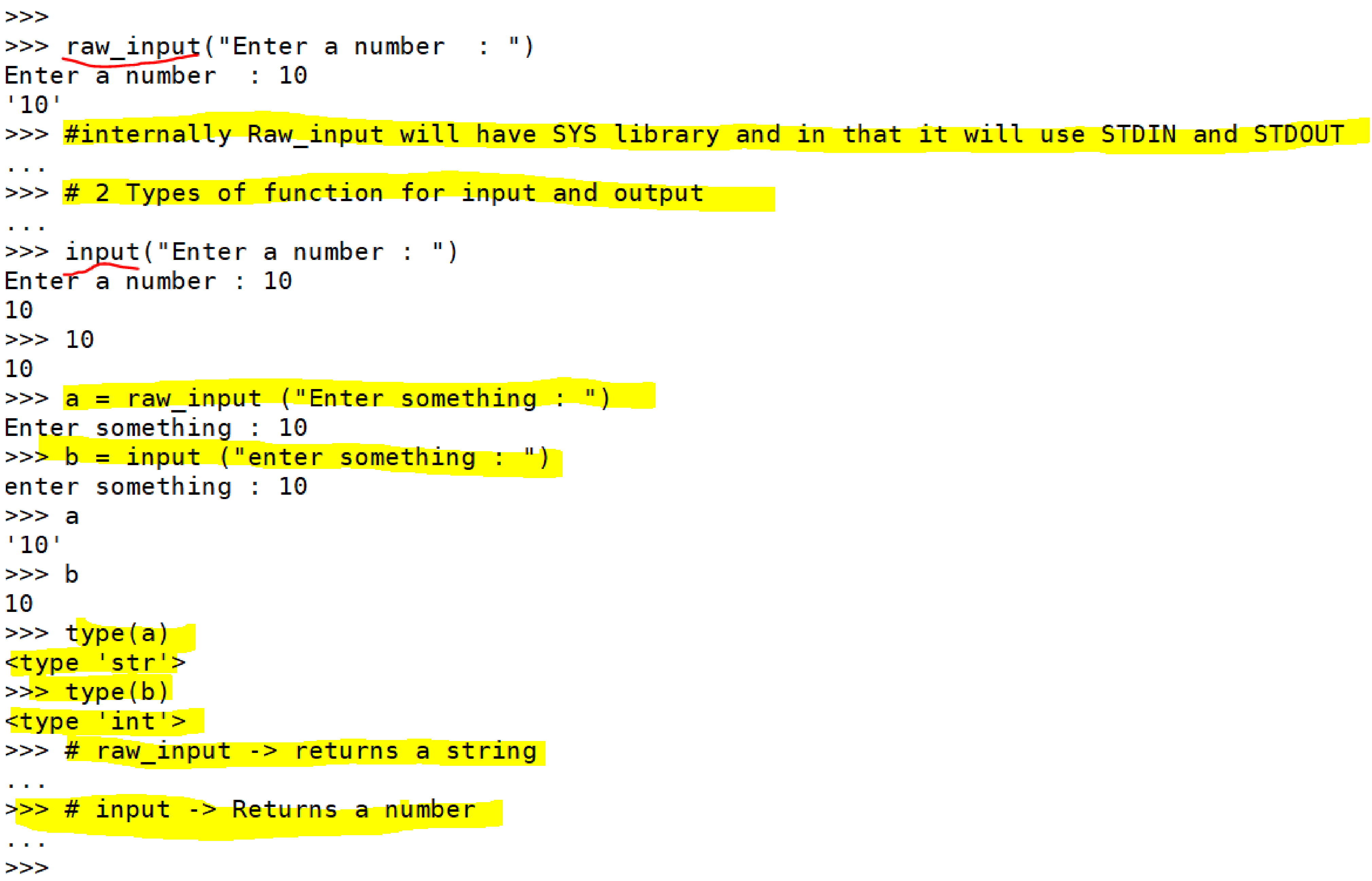
```
>>> for i in xrange(10):
        print i



0
1
2
3
4
5
6
7
8
9
>>>
```

Getting User Input from Keyboard

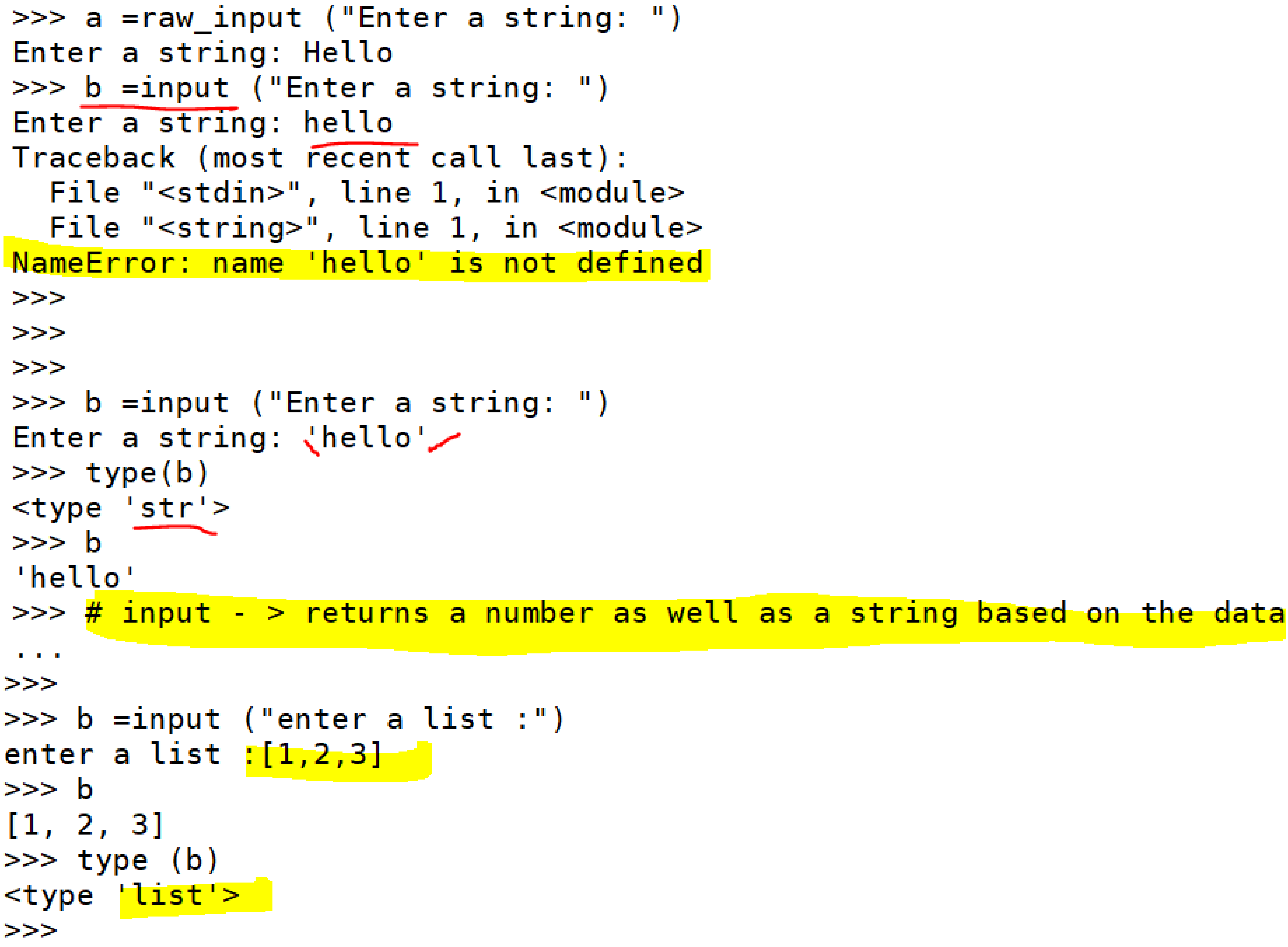
- The function **input()** can be used to read data from the user:

- You can store the result into a variable

- ***Raw_input*** is the function which help you to interact with keybord



```
>>>
>>> raw_input("Enter a number  : ")
Enter a number  : 10
'10'
>>> #internally Raw_input will have SYS library and in that it will use STDIN and STDOUT
...
>>> # 2 Types of function for input and output
...
>>> input("Enter a number : ")
Enter a number : 10
10
>>> 10
10
>>> a = raw_input ("Enter something :  ")
Enter something : 10
>>> b = input ("enter something : ")
enter something : 10
>>> a
'10'
>>> b
10
>>> type(a)
<type 'str'>
>>> type(b)
<type 'int'>
>>> # raw_input -> returns a string
...
>>> # input -> Returns a number
...
>>>
```



```
>>> a =raw_input ("Enter a string: ")
Enter a string: Hello
>>> b =input ("Enter a string: ")
Enter a string: hello
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'hello' is not defined
>>>
>>>
>>>
>>> b =input ("Enter a string: ")
Enter a string: 'hello'
>>> type(b)
<type 'str'>
>>> b
'hello'
>>> # input - > returns a number as well as a string based on the data
...
>>>
>>> b =input ("enter a list :")
enter a list :[1,2,3]
>>> b
[1, 2, 3]
>>> type (b)
<type 'list'>
>>>
```

Flow Control

- Python provides various tools for flow control
- Some of them are:
  - If
  - If else
  - While
  - For
  - Pass
  - Break
  - Continue

Break and Continue Statement

- **Break** and **Continue** statements are used to exit from loop

- The **break statement** is used to **break** out of loop statement i.e. stop the execution of a looping statement, even if the loop condition has not become false or the sequence of items has not been completely iterated over

- The **continue statement** is used to tell Python to skip the rest of the statements in the current loop block and to **continue** to the next iteration of the loop.

Pass Statement

- The **pass statement** does nothing. It can be used when a statement is required syntactically but the program requires no action.
- In simpler words, you can not leave a statement empty in Python. In this situation you can place statement there.
- Used commonly while creating minimal classes.
- ***Syntax :***

```
While True
Pass
Class MyEmptyClass
Pass
```

If – else Statement

- This **if** statement is used to check a condition. If the condition is true, we run a block of statements (called the **if-block**), else we process another block of statements (called the **else block**)
- The **else** clause is optional

Note: Keep a check on indentation and do not forget the colon (:)

Syntax :

```
If (condition):
    Statements…
Else:
    Default option statements…
```

While Statement

- The **while statement** allows you to repeatedly execute a block of statements as long as a condition is true.
- Indentation and Colon should be respected.

Syntax :

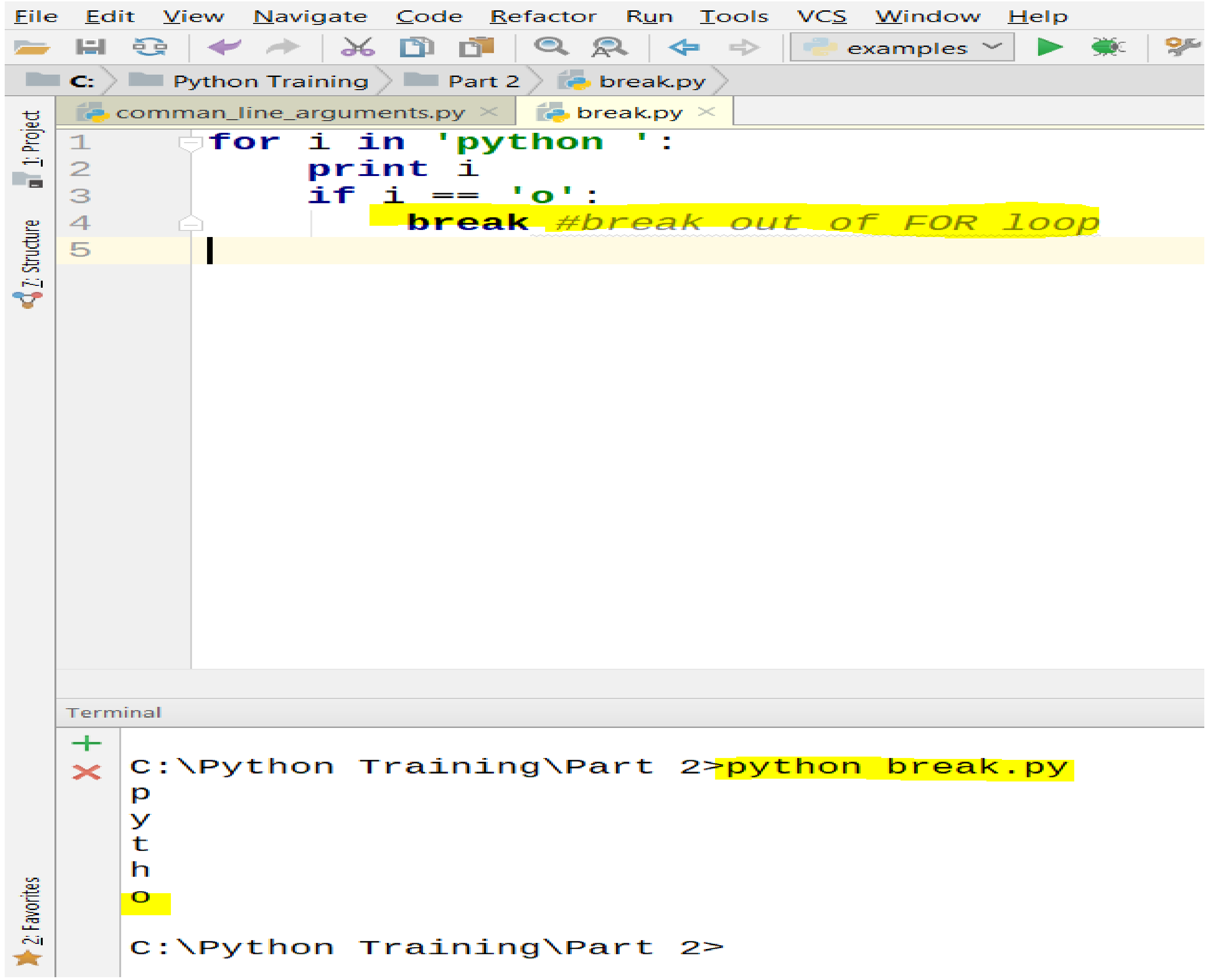
```
    While Expression
    Statement(s)
```

For statement

- The **for...in** statement is another looping statement which **iterates** over a sequence of objects i.e. go through each item in a sequence.

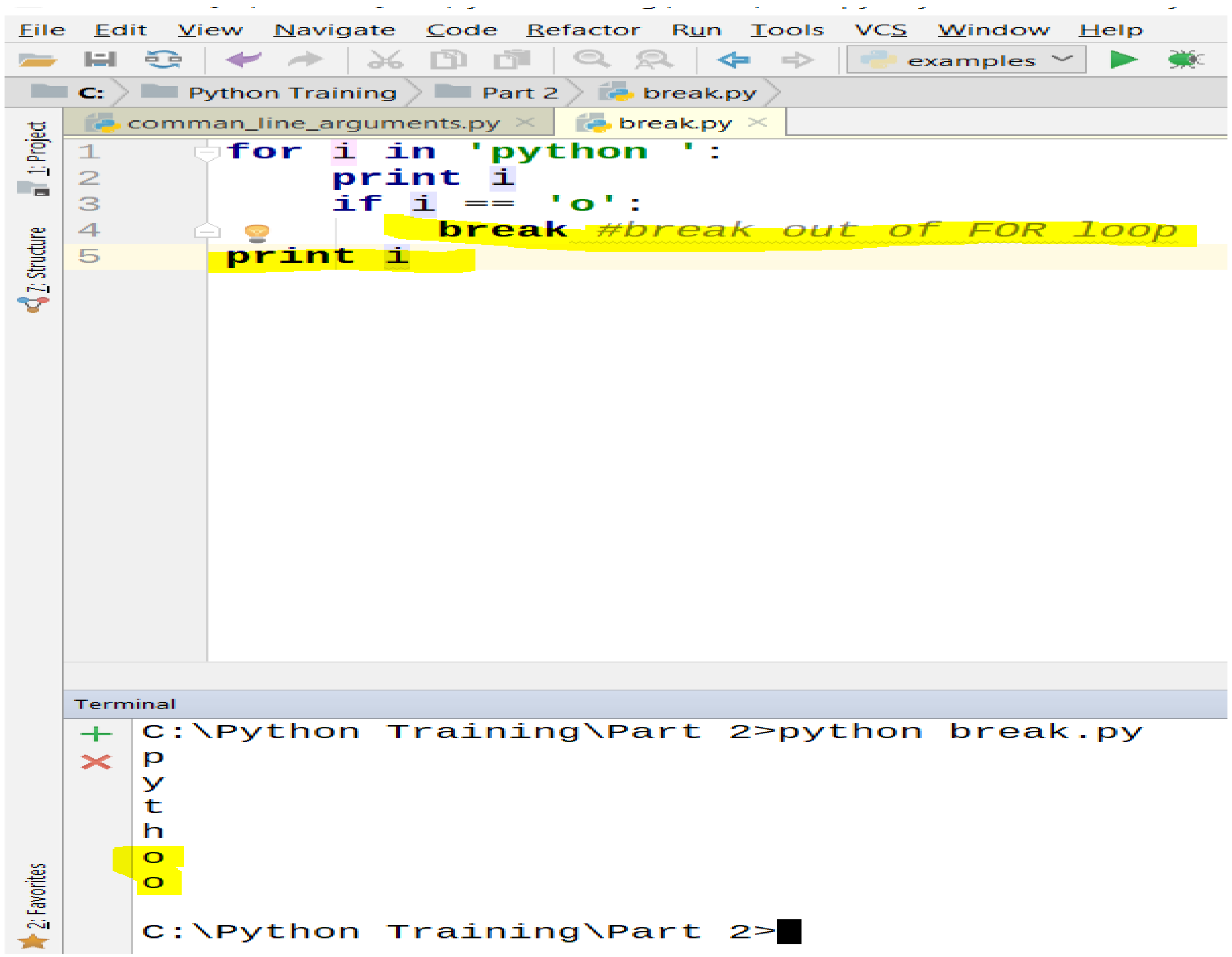
Syntax :

```
For iterator_name in
iterating_sequence:
…statements…
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
examples
C: Python Training Part 2 break.py
comman_line_arguments.py
break.py
1: Project
7: Structure
2: Favorites
Terminal

```python
for i in 'python ':
    print i
    if i == 'o':
        break #break out of FOR loop
```

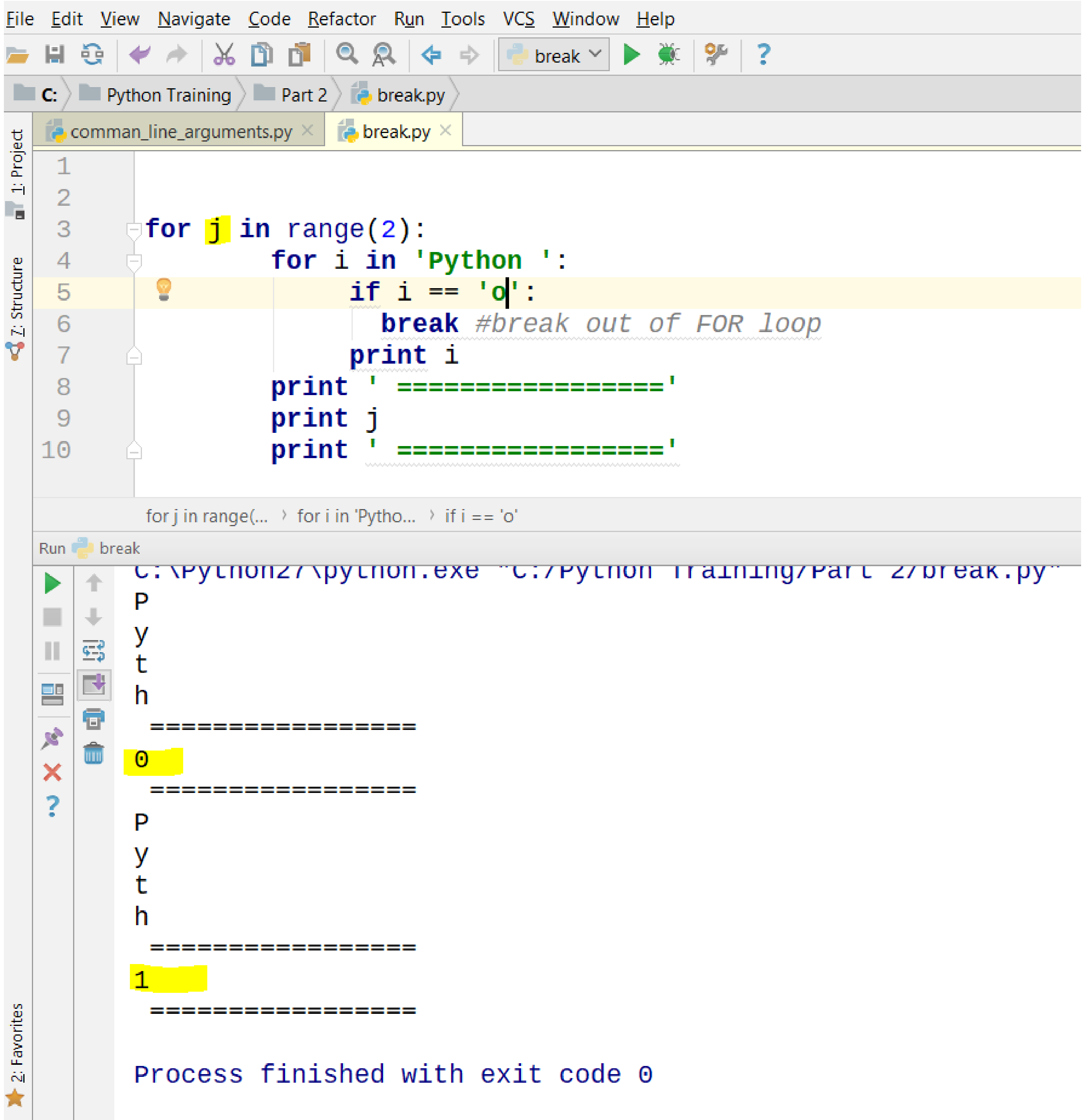
```
C:\Python Training\Part 2>python break.py
p
y
t
h
o

C:\Python Training\Part 2>
```



```
for i in 'python ':
    print i
    if i == 'o':
        break #break out of FOR loop
print i
```

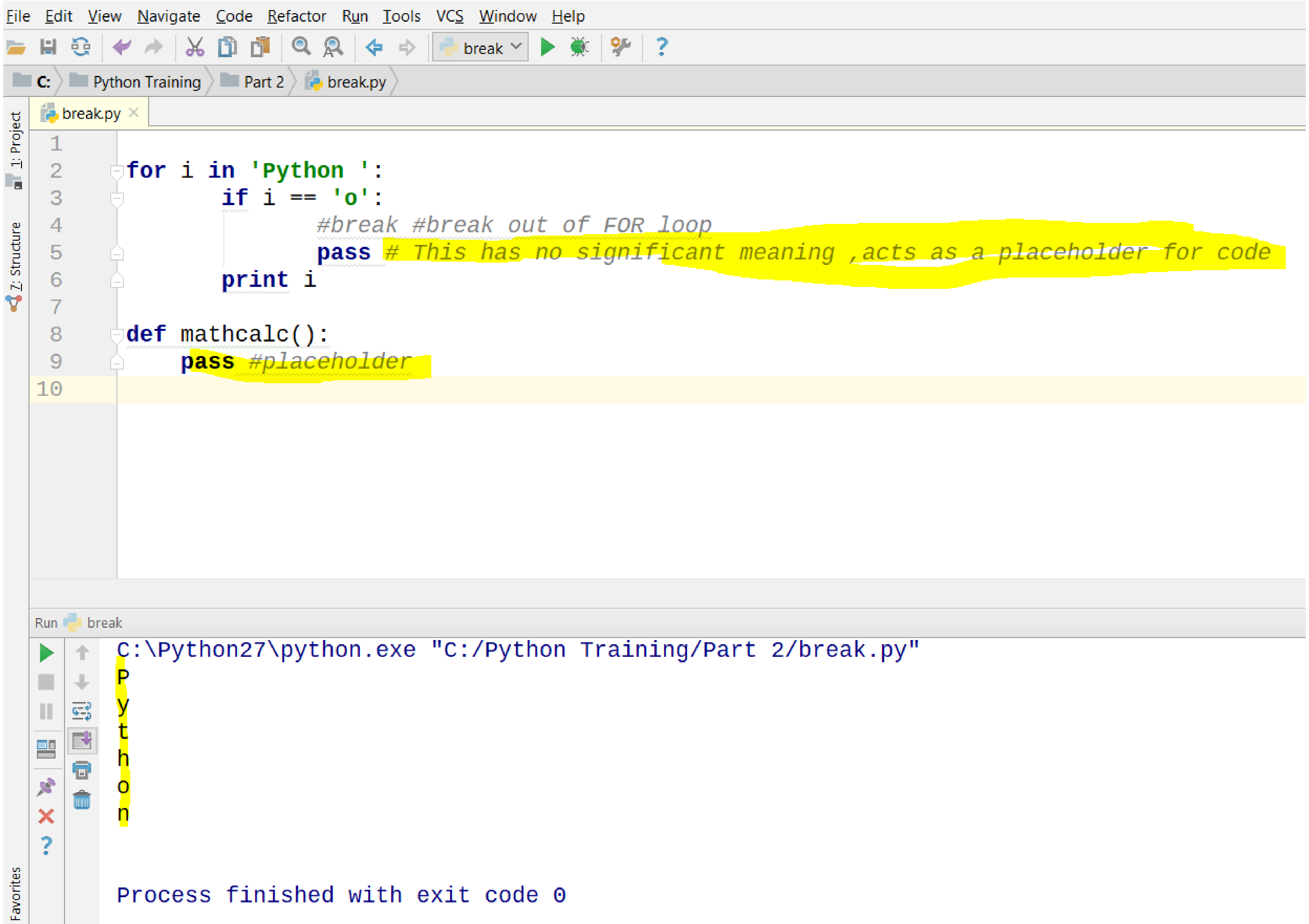
```
C:\Python Training\Part 2>python break.py
p
y
t
h
o
o

C:\Python Training\Part 2>
```

- Break goes out of the loop.


File Edit View Navigate Code Refactor Run Tools VCS Window Help
break
C: › Python Training › Part 2 › break.py
comman_line_arguments.py
break.py
1: Project
7: Structure
2: Favorites
for j in range(... › for i in 'Pytho... › if i == 'o'
Run break

```
for j in range(2):
        for i in 'Python ':
             if i == 'o':
               break #break out of FOR loop
             print i
        print ' ================='
        print j
        print ' ================='
```

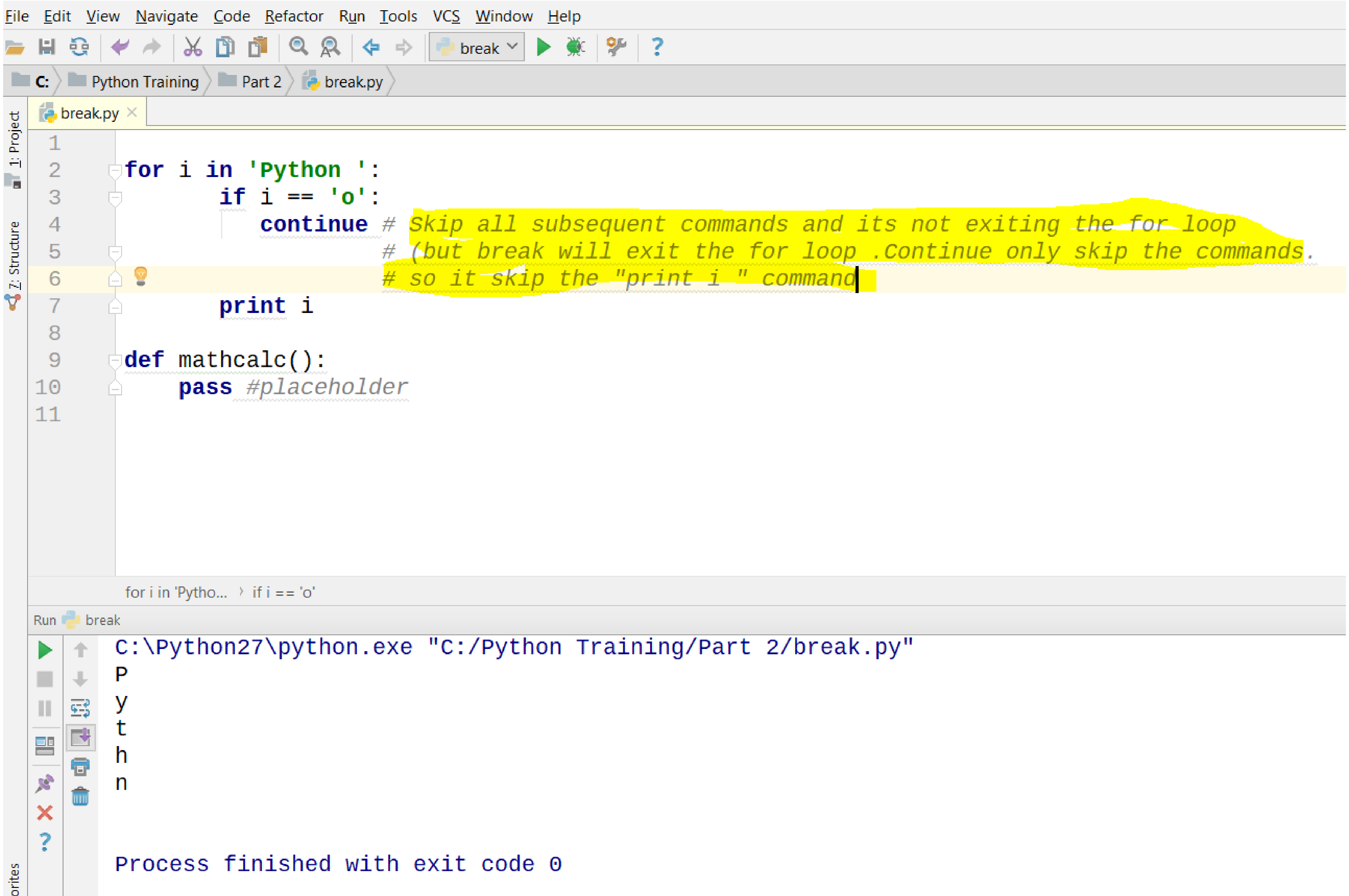
```
C:\Python27\python.exe "C:/Python Training/Part 2/break.py"
P
y
t
h
 =================
0
 =================
P
y
t
h
 =================
1
 =================

Process finished with exit code 0
```



File Edit View Navigate Code Refactor Run Tools VCS Window Help
break
C: Python Training Part 2 break.py
break.py
1: Project
7: Structure
Favorites
Run break

```
for i in 'Python ':
        if i == 'o':
                #break #break out of FOR loop
                pass # This has no significant meaning ,acts as a placeholder for code
        print i

def mathcalc():
    pass #placeholder
```

```
C:\Python27\python.exe "C:/Python Training/Part 2/break.py"
P
y
t
h
o
n


Process finished with exit code 0
```



File Edit View Navigate Code Refactor Run Tools VCS Window Help
break
C: Python Training Part 2 break.py
break.py
1: Project
7: Structure
for i in 'Pytho... › if i == 'o'
Run break
orites

```
for i in 'Python ':
       if i == 'o':
          continue # Skip all subsequent commands and its not exiting the for loop
                   # (but break will exit the for loop .Continue only skip the commands.
                   # so it skip the "print i " command
       print i

def mathcalc():
    pass #placeholder
```

```
C:\Python27\python.exe "C:/Python Training/Part 2/break.py"
P
y
t
h
n


Process finished with exit code 0
```

SCDS Links

Send SCDS an Email:

scds@mcmaster.ca

Register for a Workshop:

https://u.mcmaster.ca/scds-workshops

Subscribe to our Newsletter:

https://u.mcmaster.ca/sign-up

Schedule a Consultation:

https://libcal.mcmaster.ca/appointments