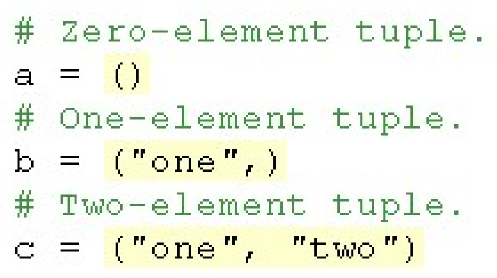
# Tuple

# Tuple

- A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
- A simple *immutable ordered sequence of items*
- Tuples are defined using parentheses (and commas).



```
# Zero-element tuple.
a = ()
# One-element tuple.
b = ("one",)
# Two-element tuple.
c = ("one", "two")
```

- Items can be of mixed types, including collection types

  **tu = (23, ‘abc’, 4.56, (2,3), ‘def’)**

## Tuple element access

- Access individual members of a tuple, list, or string using square bracket "array" notation
- *Note that all are 0 based…*

```
>>> tu = (23, 'abc', 4.56, 'spam', 'def')
>>> tu[1]     # Second item in the tuple.
 'abc'

>>> li = ["abc", 34, 4.34, 23]
>>> li[1]      # Second item in the list.
 34

>>> st = "Hello World"
>>> st[1]   # Second character in string.
 'e'
```

## Positive and negative indices

```
>>> t = (23, 'abc', 4.56, 'spam', 'def')
```

Positive index: count from the left, starting with 0

```
>>> t[1]
'abc'
```

Negative index: count from right, starting with –1

```
>>> t[-3]
4.56
```

# Nested Tuples

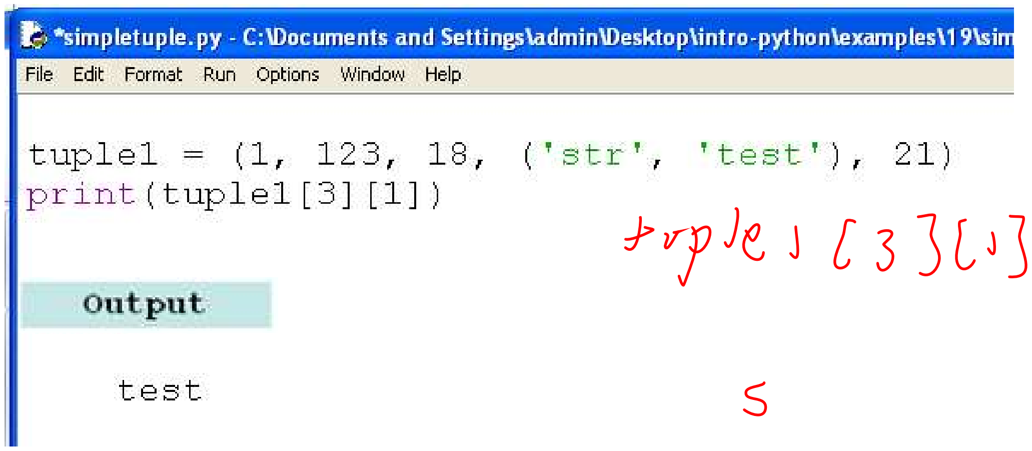
- **Tuples may be nested**

>>> t=(123, 542, 'bar')

>>> u = t, (1,2)

>>> u

**((123, 542, 'bar'), (1,2))**


*simpletuple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\19\sim
File Edit Format Run Options Window Help

```
tuple1 = (1, 123, 18, ('str', 'test'), 21)
print(tuple1[3][1])
```

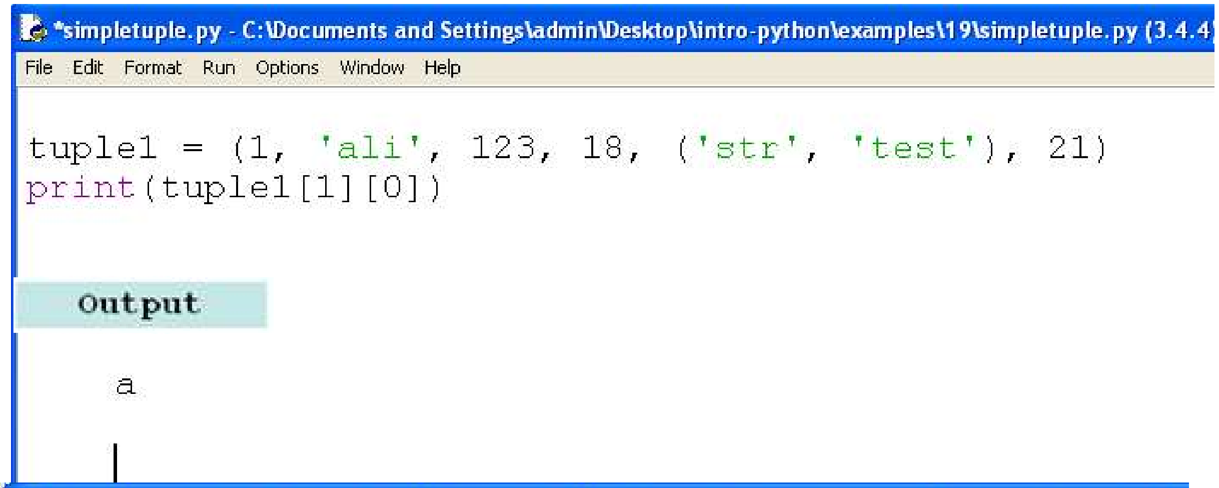
tuple1 [3][1][2]

**Output**

```
test
```

s

## Nested Tuples

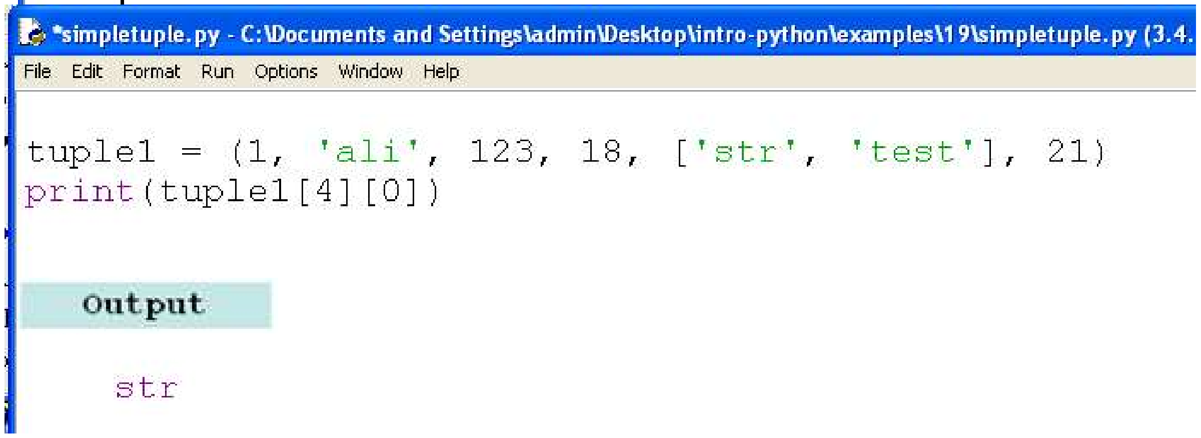

*simpletuple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\19\simpletuple.py (3.4.4)

File Edit Format Run Options Window Help

```
tuple1 = (1, 'ali', 123, 18, ('str', 'test'), 21)
print(tuple1[1][0])
```

Output

```
a
```


*simpletuple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\19\simpletuple.py (3.4.

File Edit Format Run Options Window Help

```
tuple1 = (1, 'ali', 123, 18, ['str', 'test'], 21)
print(tuple1[4][0])
```

Output

```
str
```

## Tuples are immutable

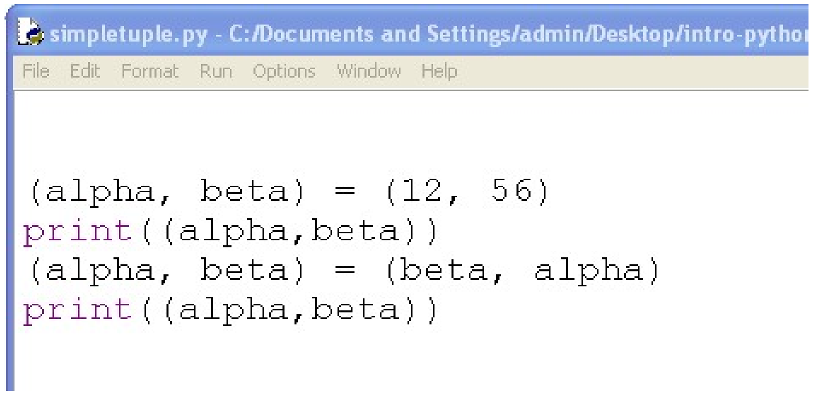
```
>>> t = (23, ‘abc’, 4.56, (2,3), ‘def’)
>>> t[2] = 3.14

Traceback (most recent call last):
  File "<pyshell#75>", line 1, in -toplevel-
    tu[2] = 3.14
TypeError: object doesn't support item assignment
```

- You can’t change a tuple.
- You can make a fresh tuple and assign its reference to a previously used name.

```
>>> t = (23, ‘abc’, 3.14, (2,3), ‘def’)
```

- *The immutability of tuples means they’re faster than lists.*

## Swapping values


simpletuple.py - C:/Documents and Settings/admin/Desktop/intro-pytho
File Edit Format Run Options Window Help

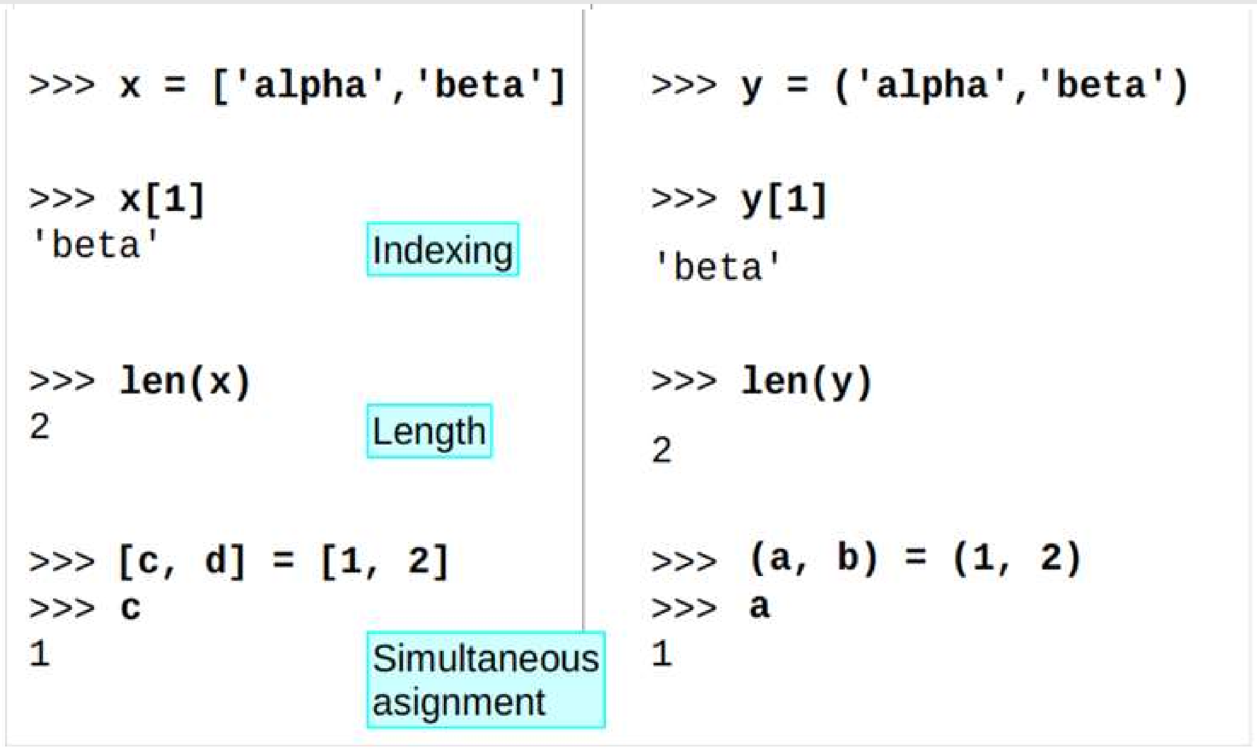
```
(alpha, beta) = (12, 56)
print((alpha,beta))
(alpha, beta) = (beta, alpha)
print((alpha,beta))
```

a s (a b [illegible]

**Output**

```
 RESTART: C:/Documents and Settings/adm
etuple.py
(12, 56)
(56, 12)
```

## Tuples and lists: similarities

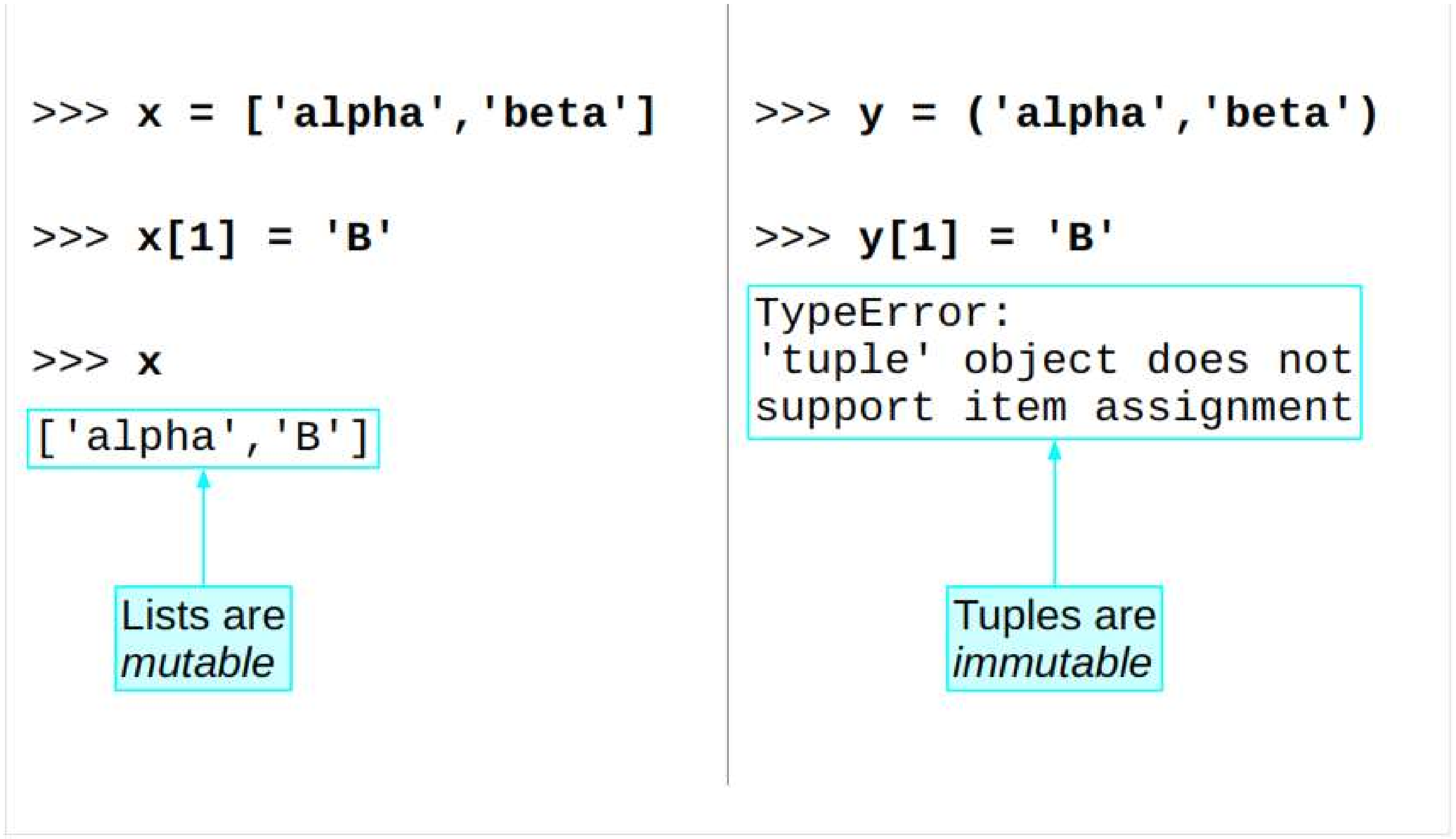


```
>>> x = ['alpha','beta']

>>> x[1]
'beta'

>>> len(x)
2

>>> [c, d] = [1, 2]
>>> c
1
```

```
>>> y = ('alpha','beta')

>>> y[1]
'beta'

>>> len(y)
2

>>> (a, b) = (1, 2)
>>> a
1
```

Now that we have seen how tuples work we should take a moment to observe that they are very much like lists. So why don't we just use lists? There are very many properties that lists and tuples share.

## Tuples and lists: differences



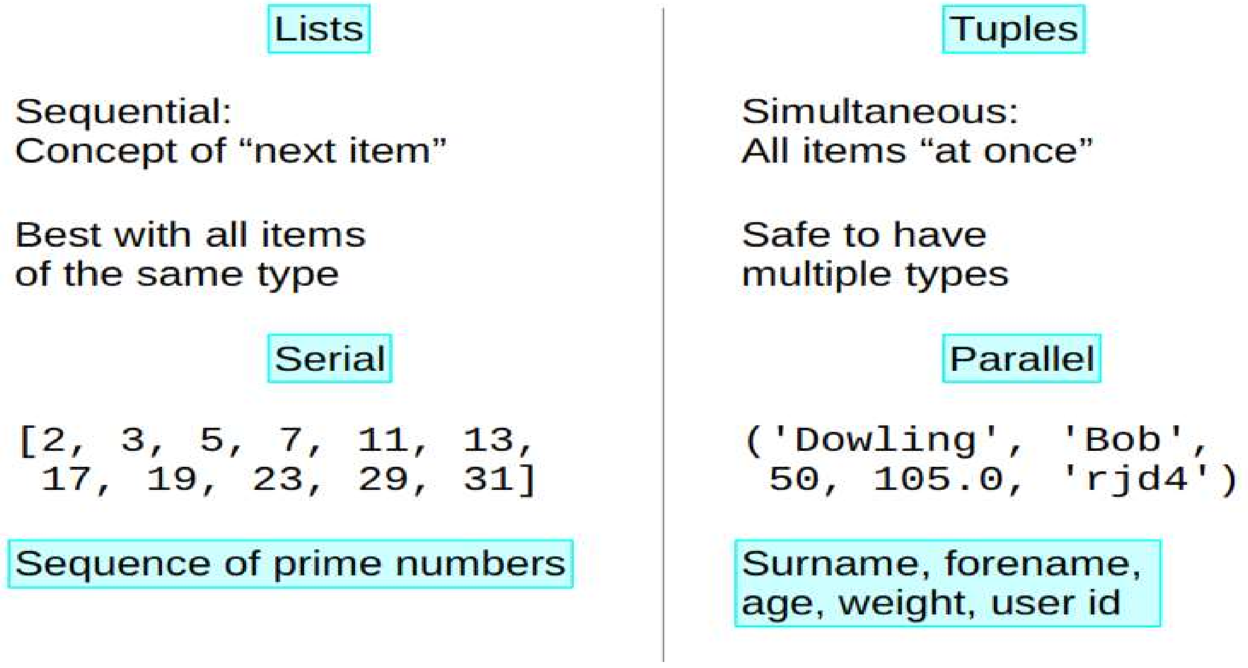
```
>>> x = ['alpha','beta']

>>> x[1] = 'B'

>>> x
['alpha','B']
```

```
>>> y = ('alpha','beta')

>>> y[1] = 'B'
TypeError:
'tuple' object does not
support item assignment
```

But there is one critical difference. Like strings, tuples are immutable.

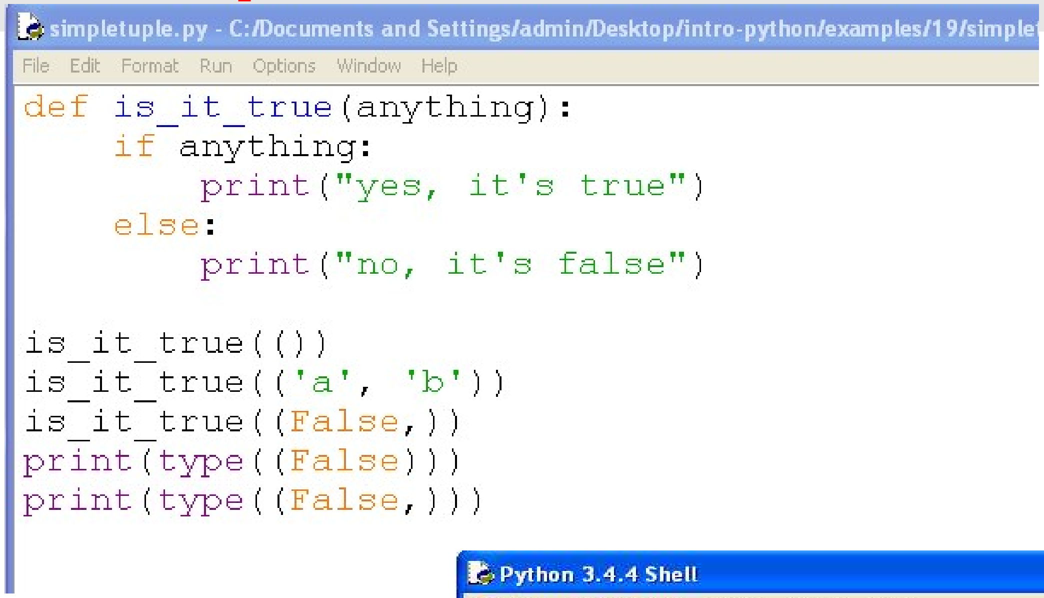
## Tuples and lists: philosophy

- Philosophically, it is appropriate to use lists where there is a sequence of items or where you *need mutability. Tuples are more appropriate for* circumstances where you think about all the items at once.
- Because of this, tuples are appropriate where you want to join together a collection of parameters of different types into a single object.

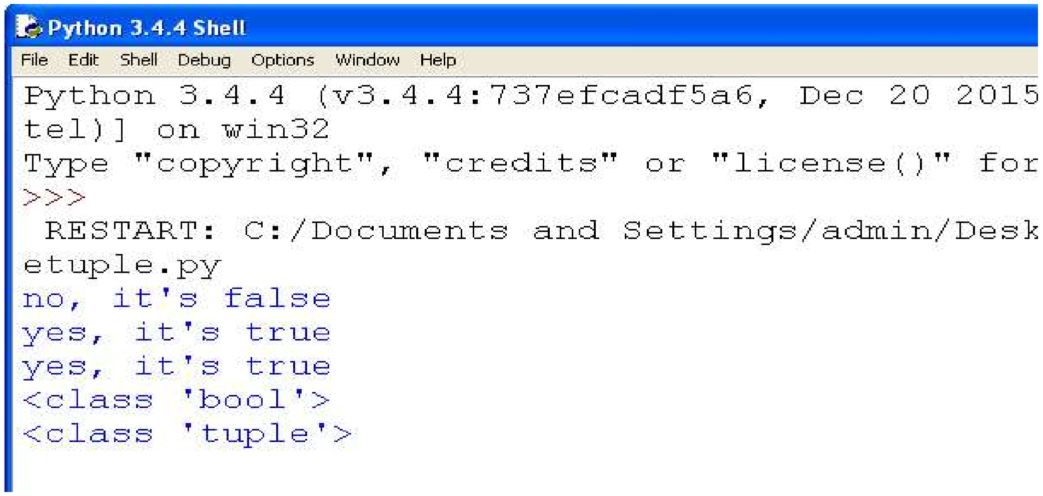


| Lists | Tuples |
|---|---|
| Sequential: Concept of "next item" | Simultaneous: All items "at once" |
| Best with all items of the same type | Safe to have multiple types |
| Serial | Parallel |
| `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]` | `('Dowling', 'Bob', 50, 105.0, 'rjd4')` |
| Sequence of prime numbers | Surname, forename, age, weight, user id |

## Tuples in Boolean context


simpletuple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/19/simplet
File Edit Format Run Options Window Help

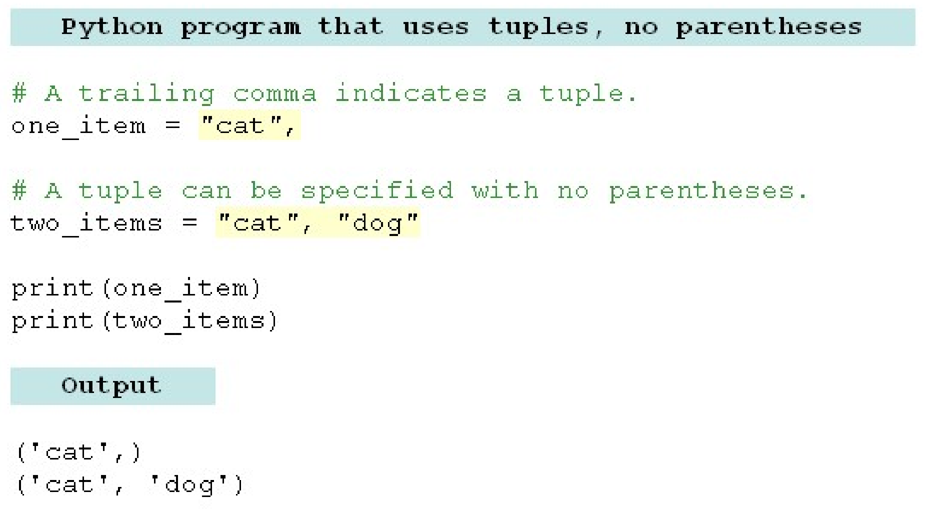
```
def is_it_true(anything):
    if anything:
        print("yes, it's true")
    else:
        print("no, it's false")

is_it_true(())
is_it_true(('a', 'b'))
is_it_true((False,))
print(type((False)))
print(type((False,)))
```



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015
tel)] on win32
Type "copyright", "credits" or "license()" for
>>>
 RESTART: C:/Documents and Settings/admin/Desk
etuple.py
no, it's false
yes, it's true
yes, it's true
<class 'bool'>
<class 'tuple'>
```

## No parentheses

- Tuples are typically specified with surrounding parentheses chars. But suppose you are a wild one. You can just use a comma. The tuple is inferred.



**Python program that uses tuples, no parentheses**

```python
# A trailing comma indicates a tuple.
one_item = "cat",

# A tuple can be specified with no parentheses.
two_items = "cat", "dog"

print(one_item)
print(two_items)
```

**Output**

```
('cat',)
('cat', 'dog')
```

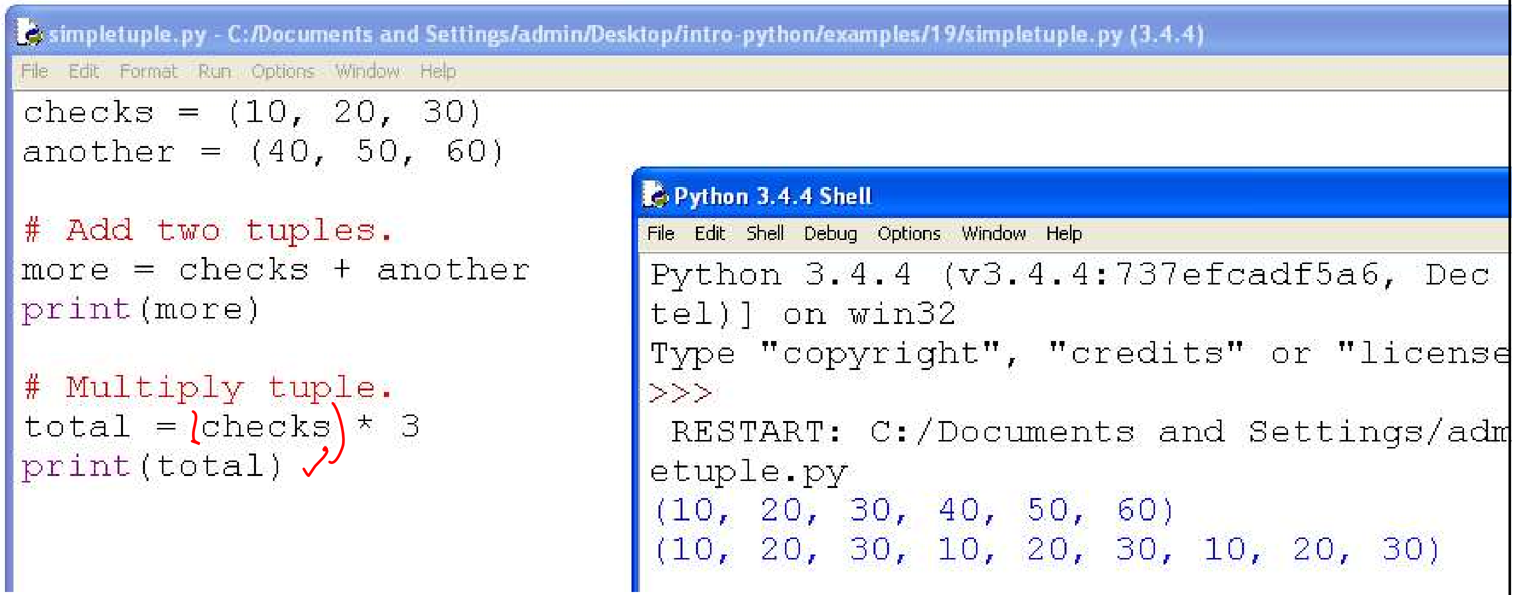
## Tuple Operators

- Tuples support operators + , * , in , not in, slicing, pack and unpack

## Tuples add and multiply

- Tuples respond to + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

# Tuples add and multiply


simpletuple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/19/simpletuple.py (3.4.4)
File Edit Format Run Options Window Help

```
checks = (10, 20, 30)
another = (40, 50, 60)

# Add two tuples.
more = checks + another
print(more)

# Multiply tuple.
total = (checks) * 3
print(total) ✓
```

Python 3.4.4 Shell
File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec
tel)] on win32
Type "copyright", "credits" or "license
>>>
 RESTART: C:/Documents and Settings/adm
etuple.py
(10, 20, 30, 40, 50, 60)
(10, 20, 30, 10, 20, 30, 10, 20, 30)
```

L = [1, 2]

M = L * 3

M = [L] * 3

[1, 2, 1, 2, 1, 2]

[[1, 2], [1, 2], [1, 2]]

((10, 20, 30), (10, 20, 30), (10, 20, 30))

## Slicing: return copy of a subset

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying *before* second.

```
>>> t[1:4]
('abc', 4.56, (2,3))
```

Negative indices count from end

```
>>> t[1:-1]
('abc', 4.56, (2,3))
```

## Slicing: return copy of a subset

```
>>> t = (23, ‘abc’, 4.56, (2,3), ‘def’)
```

Omit first index to make copy starting from beginning of the container

```
    >>> t[:2]
    (23, ‘abc’)
```

Omit second index to make copy starting at first index and going to end

```
    >>> t[2:]
    (4.56, (2,3), ‘def’)
```

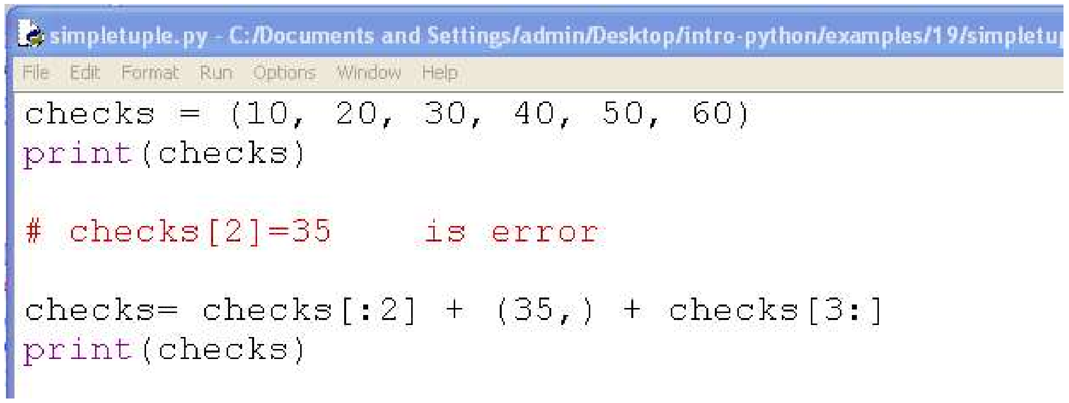
## Copying the Whole Sequence

- [ : ] makes a *copy* of an entire sequence

```
t= (23, 'abc', 4.56, (2,3), 'def')
m=t[:]
print(m)

(23, 'abc', 4.56, (2,3), 'def')
```

-

# Changing elements


simpletuple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/19/simpletu
File Edit Format Run Options Window Help

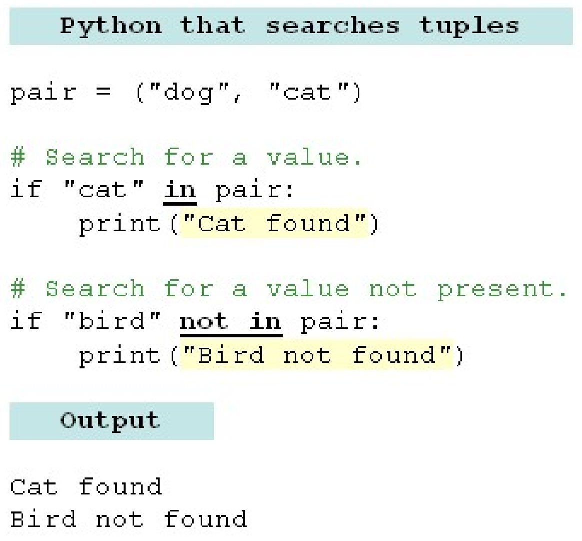
```python
checks = (10, 20, 30, 40, 50, 60)
print(checks)

# checks[2]=35    is error

checks= checks[:2] + (35,) + checks[3:]
print(checks)
```

**Output**

```
(10, 20, 30, 40, 50, 60)
(10, 20, 35, 40, 50, 60)
```

## Tuples In keyword

- This example creates a two-element tuple (a pair). It searches the tuple for the string "cat". It then searches for "bird", but this string does not exist.



**Python that searches tuples**

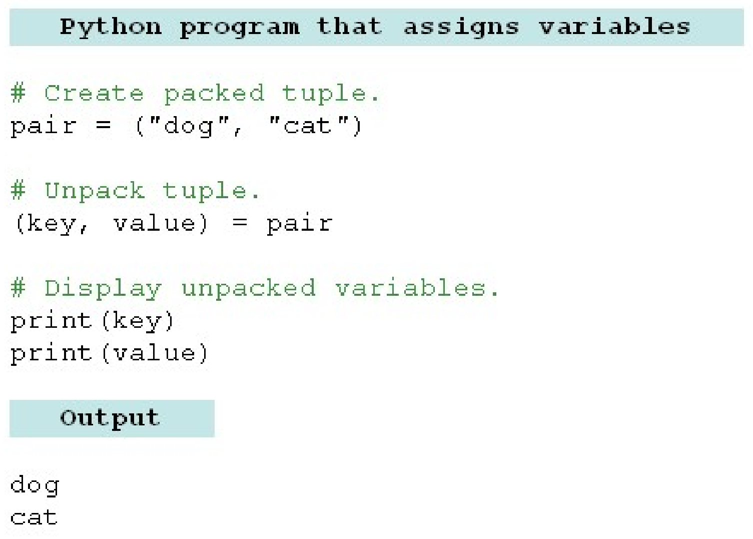
```
pair = ("dog", "cat")

# Search for a value.
if "cat" in pair:
    print("Cat found")

# Search for a value not present.
if "bird" not in pair:
    print("Bird not found")
```

**Output**

```
Cat found
Bird not found
```

## Pack, unpack

- Tuples can be packed and unpacked. In packing, we place values into a new tuple. And in unpacking we extract those values back into variables.



**Python program that assigns variables**

```
# Create packed tuple.
pair = ("dog", "cat")

# Unpack tuple.
(key, value) = pair

# Display unpacked variables.
print(key)
print(value)
```

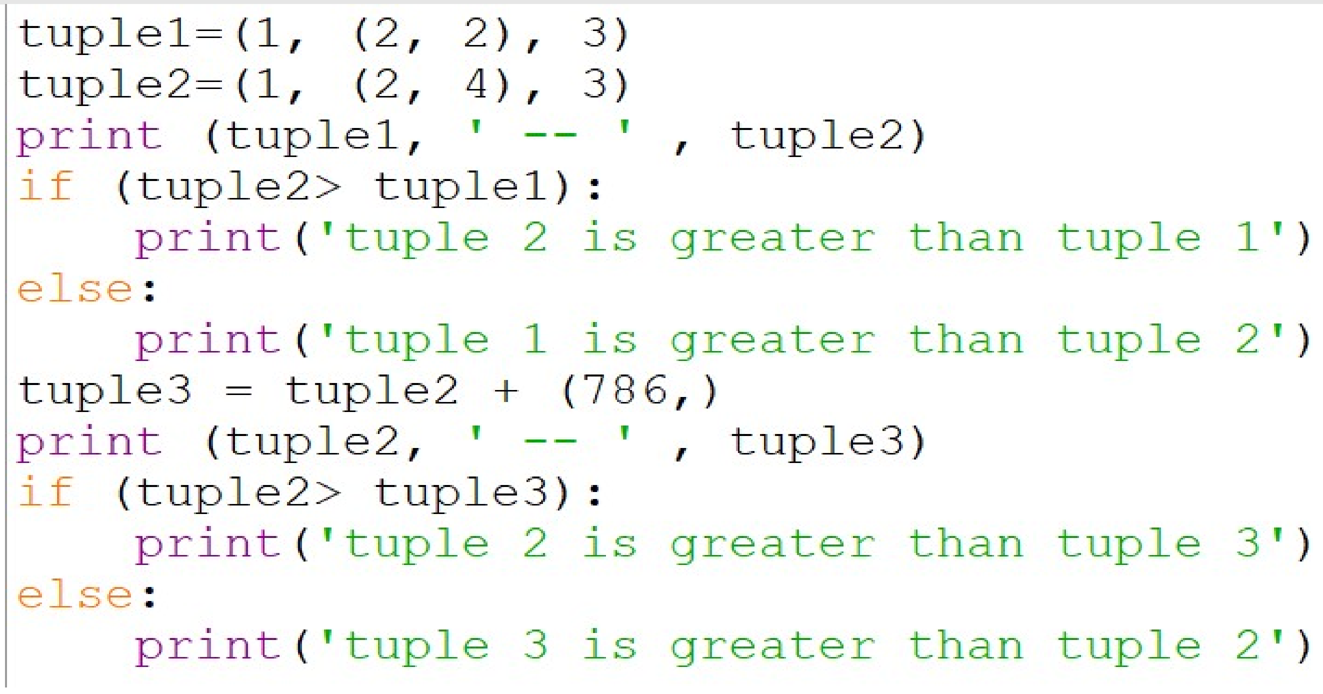
**Output**

```
dog
cat
```

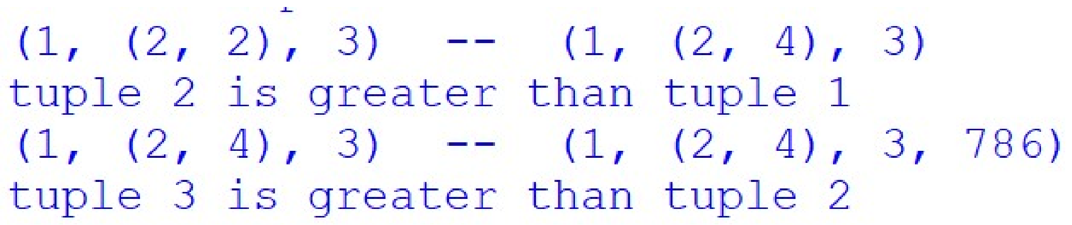
## Tuple Comparison

- The operators  >, <, ==  , !=, <=. >= compares elements of two tuples.

## Tuple Comparison



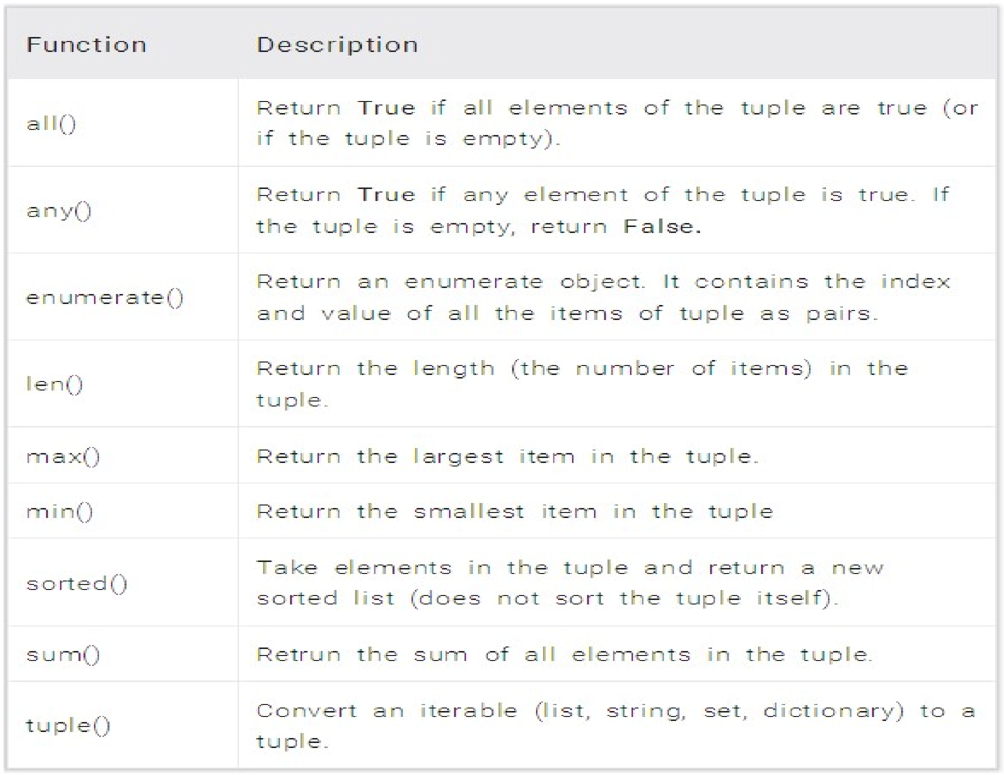
```
tuple1=(1, (2, 2), 3)
tuple2=(1, (2, 4), 3)
print (tuple1, ' -- ' , tuple2)
if (tuple2> tuple1):
    print('tuple 2 is greater than tuple 1')
else:
    print('tuple 1 is greater than tuple 2')
tuple3 = tuple2 + (786,)
print (tuple2, ' -- ' , tuple3)
if (tuple2> tuple3):
    print('tuple 2 is greater than tuple 3')
else:
    print('tuple 3 is greater than tuple 2')
```



```
(1, (2, 2), 3)  --  (1, (2, 4), 3)
tuple 2 is greater than tuple 1
(1, (2, 4), 3)  --  (1, (2, 4), 3, 786)
tuple 3 is greater than tuple 2
```

# Built-in Functions on Tuples

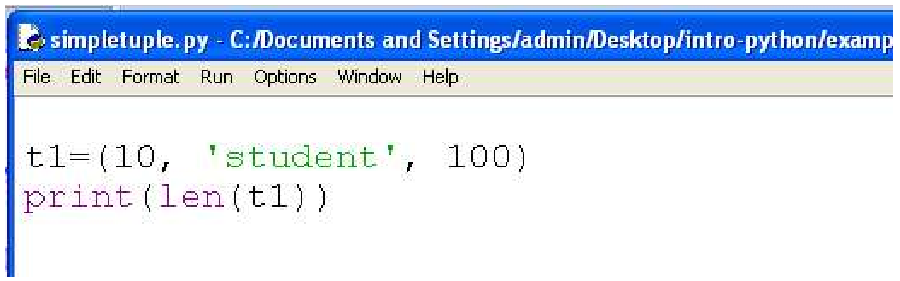
- Tuples respond to all of the general sequence operations we used on strings in the prior chapter



| Function | Description |
|---|---|
| all() | Return **True** if all elements of the tuple are true (or if the tuple is empty). |
| any() | Return **True** if any element of the tuple is true. If the tuple is empty, return **False.** |
| enumerate() | Return an enumerate object. It contains the index and value of all the items of tuple as pairs. |
| len() | Return the length (the number of items) in the tuple. |
| max() | Return the largest item in the tuple. |
| min() | Return the smallest item in the tuple |
| sorted() | Take elements in the tuple and return a new sorted list (does not sort the tuple itself). |
| sum() | Retrun the sum of all elements in the tuple. |
| tuple() | Convert an iterable (list, string, set, dictionary) to a tuple. |

# Len on Tuples

- Len function return the length of a tuple


simpletuple.py - C:/Documents and Settings/admin/Desktop/intro-python/examp
File Edit Format Run Options Window Help

```
t1=(10, 'student', 100)
print(len(t1))
```

## Min and max

- The max and min functions can be used on tuples. These functions locate the item that would be sorted last (max) or sorted first (min).

**Python program that uses max and min**

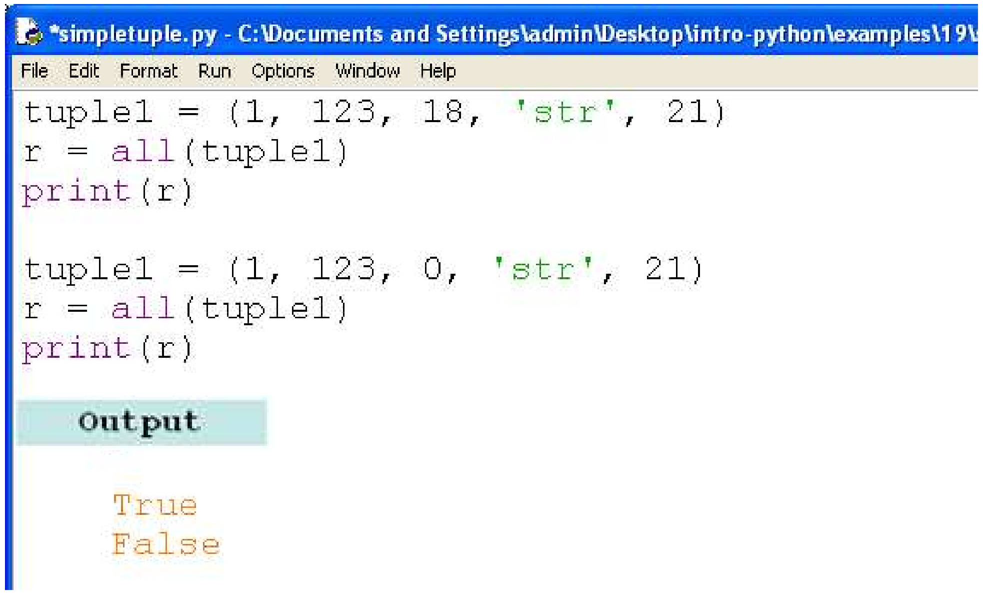
```
# Max and min for strings.
friends = ("sandy", "michael", "aaron", "stacy")

print(max(friends))
print(min(friends))

# Max and min for numbers.
earnings = (1000, 2000, 500, 4000)

print(max(earnings))
print(min(earnings))
```

**Output**

```
stacy
aaron
4000
500
```

## All

- Return True if all elements of the tuple are true (or if the tuple is empty).


*simpletuple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\19\
File Edit Format Run Options Window Help

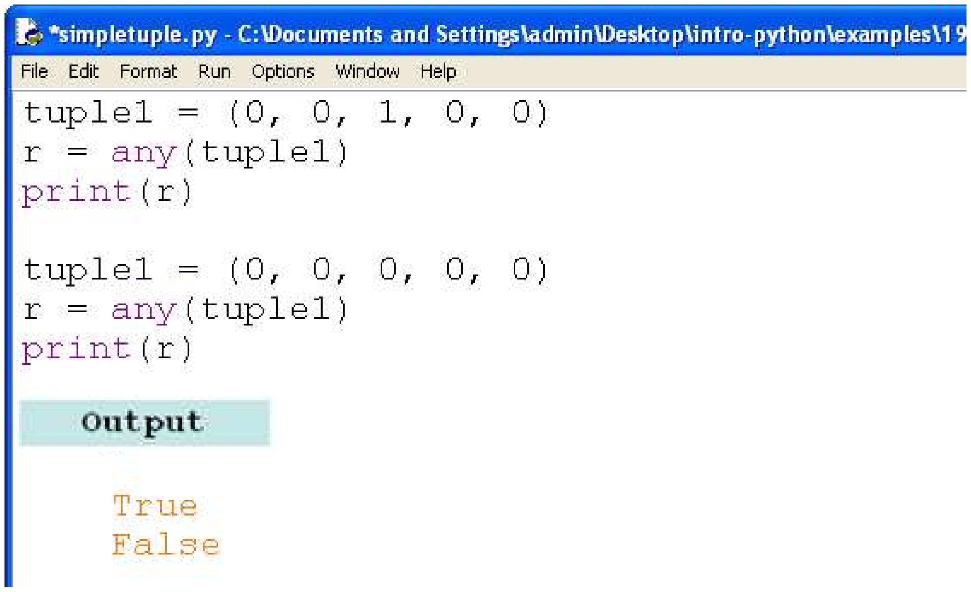
```
tuple1 = (1, 123, 18, 'str', 21)
r = all(tuple1)
print(r)

tuple1 = (1, 123, 0, 'str', 21)
r = all(tuple1)
print(r)
```

Output

```
True
False
```

## Any

- Return True if any element of the tuple is true. If the tuple is empty, return False.


*simpletuple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\19
File Edit Format Run Options Window Help

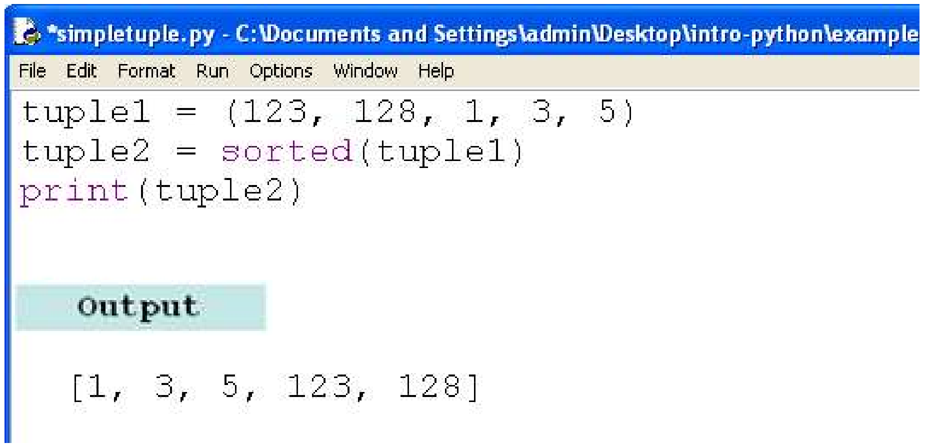
```
tuple1 = (0, 0, 1, 0, 0)
r = any(tuple1)
print(r)

tuple1 = (0, 0, 0, 0, 0)
r = any(tuple1)
print(r)
```

Output

```
True
False
```

## Sorted

- Take elements in the tuple and return a new sorted list (does not sort the tuple itself).


*simpletuple.py - C:\Documents and Settings\admin\Desktop\intro-python\example
File Edit Format Run Options Window Help

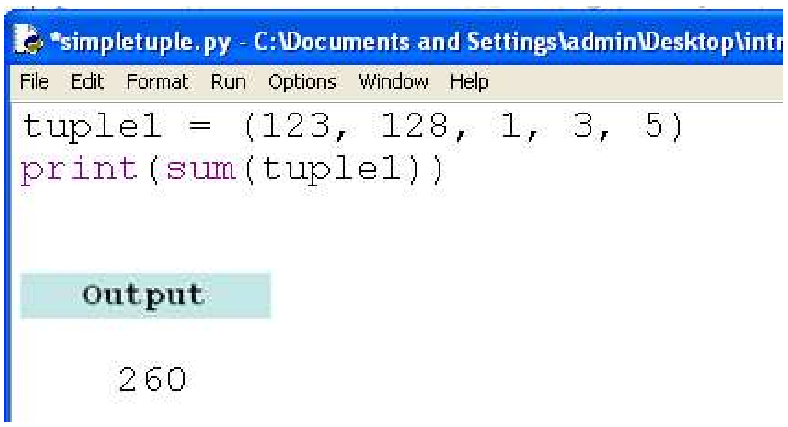
```
tuple1 = (123, 128, 1, 3, 5)
tuple2 = sorted(tuple1)
print(tuple2)
```

Output

```
[1, 3, 5, 123, 128]
```

## Sum

- Retrun the sum of all elements in the tuple.


*simpletuple.py - C:\Documents and Settings\admin\Desktop\intr
File Edit Format Run Options Window Help

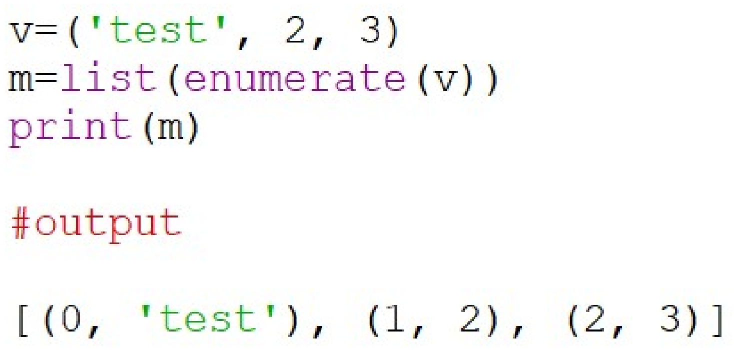
```
tuple1 = (123, 128, 1, 3, 5)
print(sum(tuple1))
```

Output

```
260
```

## Enumerate

- This is a built-in method. Enumerate() returns a tuple of an index and the element value at that index. It is often used on a list.



```
v=('test', 2, 3)
m=list(enumerate(v))
print(m)

#output

[(0, 'test'), (1, 2), (2, 3)]
```

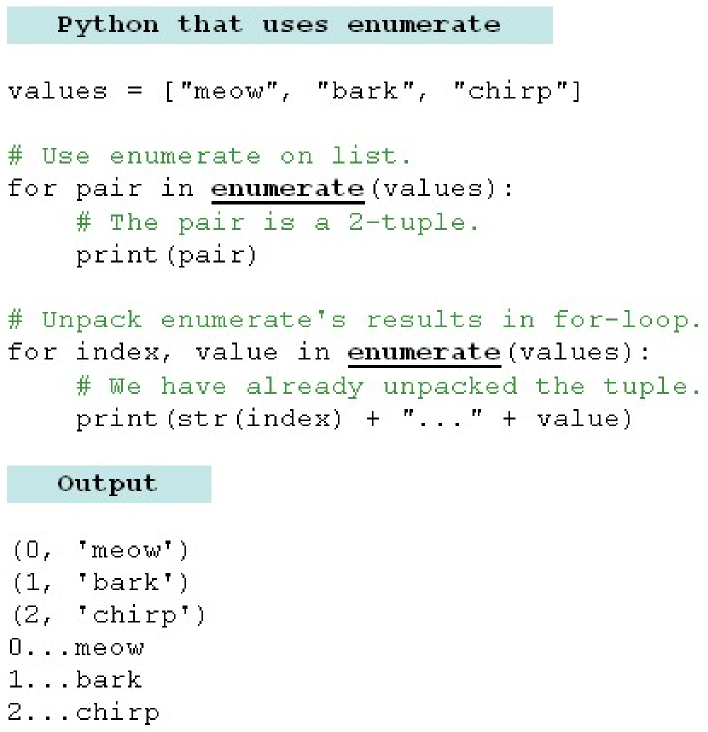
## Enumerate

- This is a built-in method. Enumerate() returns a tuple of an index and the element value at that index. It is often used on a list.

```
v=['zero', 'one', 'two']
li=list(enumerate(v))
print(li)

#output:
[(0, 'zero'), (1, 'one'), (2, 'two')]
```

## Enumerate

- This is a built-in method. Enumerate() returns a tuple of an index and the element value at that index. It is often used on a list.



**Python that uses enumerate**

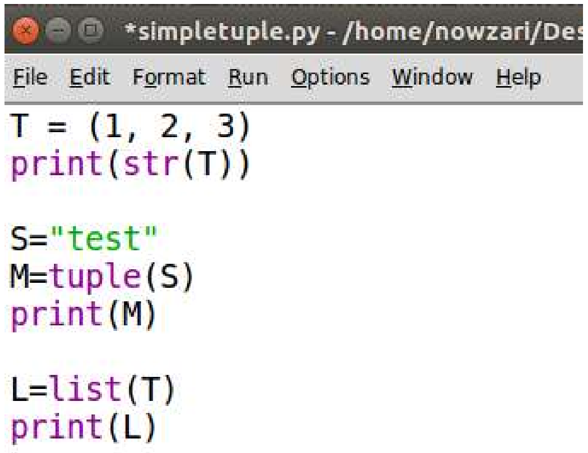
```python
values = ["meow", "bark", "chirp"]

# Use enumerate on list.
for pair in enumerate(values):
    # The pair is a 2-tuple.
    print(pair)

# Unpack enumerate's results in for-loop.
for index, value in enumerate(values):
    # We have already unpacked the tuple.
    print(str(index) + "..." + value)
```

**Output**

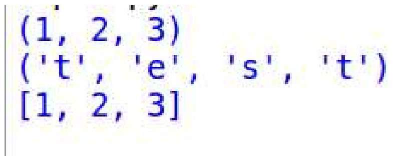
```
(0, 'meow')
(1, 'bark')
(2, 'chirp')
0...meow
1...bark
2...chirp
```

## Tuple Type conversion

- We can convert some object to tuple and  vice versa


*simpletuple.py - /home/nowzari/Des
File Edit Format Run Options Window Help

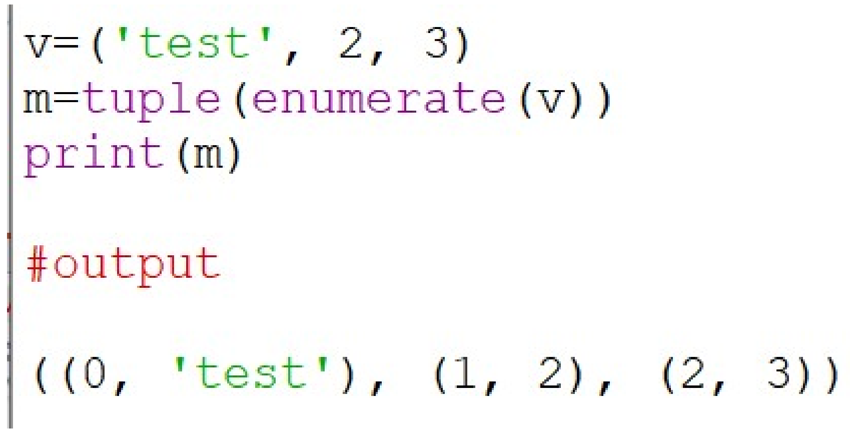
```
T = (1, 2, 3)
print(str(T))

S="test"
M=tuple(S)
print(M)

L=list(T)
print(L)
```



```
(1, 2, 3)
('t', 'e', 's', 't')
[1, 2, 3]
```

## Enumerate to tuple

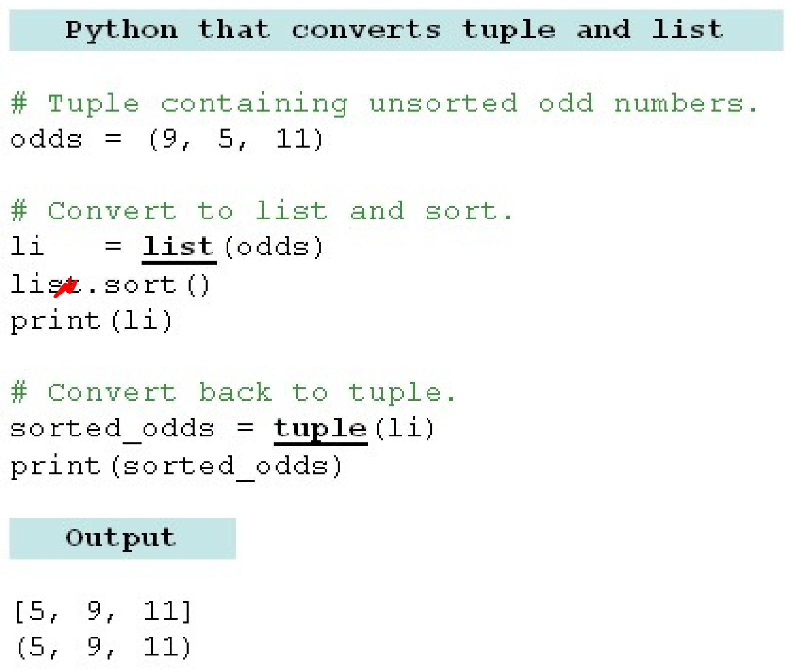
- This is a built-in method. Enumerate() returns a tuple of an index and the element value at that index. It is often used on a list.



```
v=('test', 2, 3)
m=tuple(enumerate(v))
print(m)

#output

((0, 'test'), (1, 2), (2, 3))
```

# Tuple Type conversion

- A tuple cannot be modified. But a list can be changed in many ways. For this reason we often need to convert a tuple into a list.



**Python that converts tuple and list**

```
# Tuple containing unsorted odd numbers.
odds = (9, 5, 11)

# Convert to list and sort.
li    = list(odds)
list.sort()
print(li)

# Convert back to tuple.
sorted_odds = tuple(li)
print(sorted_odds)
```

**Output**

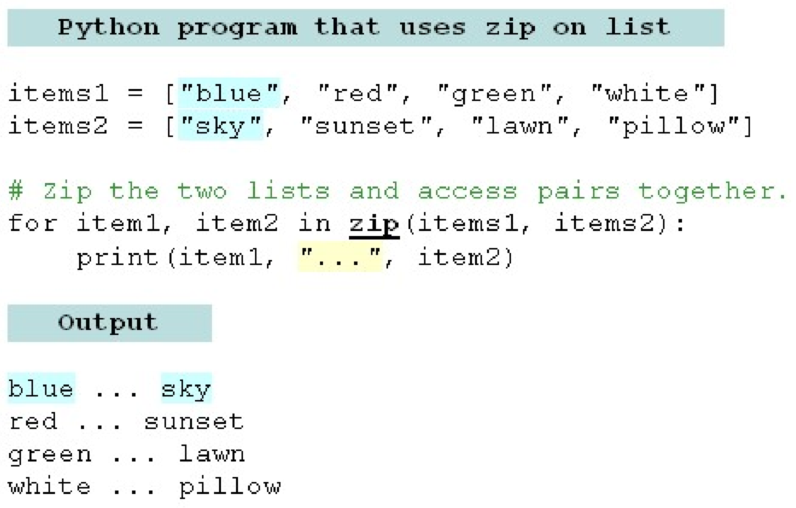
```
[5, 9, 11]
(5, 9, 11)
```

## Tuple construction with zip

- With zip we can create a list of tuples from two lists.
- Zip() is a built-in function. We pass it two iterables, like lists, and it enumerates them together.

```
items1 = ["blue", "red", "green"]
items2 = ["sky", "sunset", "lawn"]

print(list(zip(items1, items2)))

#output:
[('blue', 'sky'), ('red', 'sunset'), ('green', 'lawn')]
```

# Tuple construction with zip



**Python program that uses zip on list**

```
items1 = ["blue", "red", "green", "white"]
items2 = ["sky", "sunset", "lawn", "pillow"]

# Zip the two lists and access pairs together.
for item1, item2 in zip(items1, items2):
    print(item1, "...", item2)
```

**Output**

```
blue ... sky
red ... sunset
green ... lawn
white ... pillow
```

("blue", "sky")

## Tuple construction with zip

- We can reverse the zip operation

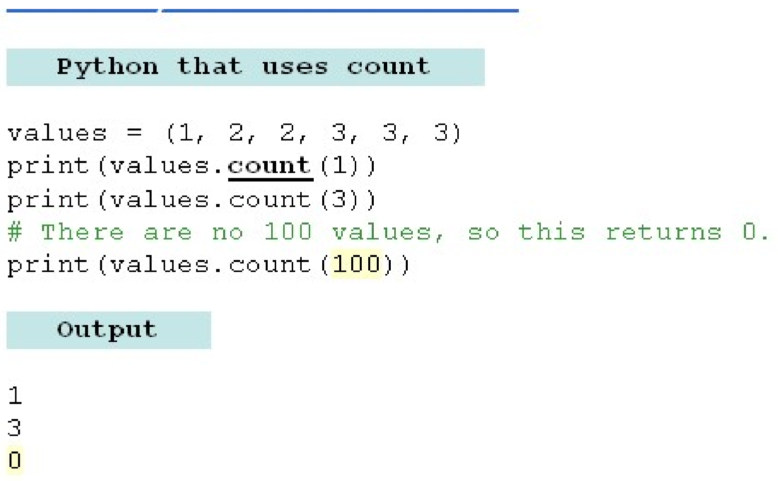
```
items1 = ["blue", "red", "green"]
items2 = ["sky", "sunset", "lawn"]
Z=list(zip(items1, items2))
print(Z)
c,d=zip(*Z)
print(c)
print(d)

#output:

[('blue', 'sky'), ('red', 'sunset'), ('green', 'lawn')]
('blue', 'red', 'green')
('sky', 'sunset', 'lawn')
```

## Type specific functions for tuples

- There are some type specific functions for tuples, such as: count, index, and type conversion methods.

## Count

- This returns the number of elements with a specific value in a tuple. If you need to get the total length of the tuple, please use len. Count only counts certain values.



Python that uses count

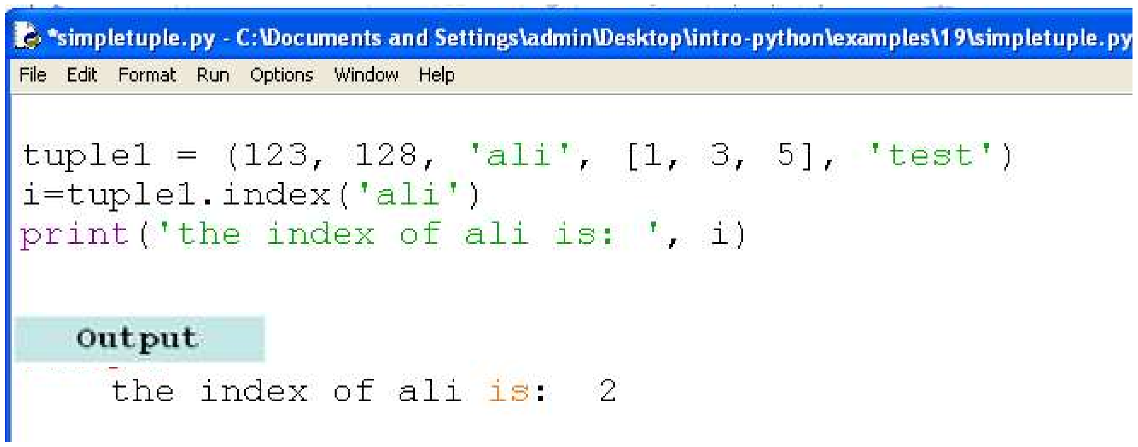
```python
values = (1, 2, 2, 3, 3, 3)
print(values.count(1))
print(values.count(3))
# There are no 100 values, so this returns 0.
print(values.count(100))
```

Output

```
1
3
0
```

## index

- Return index of first item that is equal to a specific value in a tuple


*simpletuple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\19\simpletuple.py
File Edit Format Run Options Window Help

```
tuple1 = (123, 128, 'ali', [1, 3, 5], 'test')
i=tuple1.index('ali')
print('the index of ali is: ', i)
```

Output

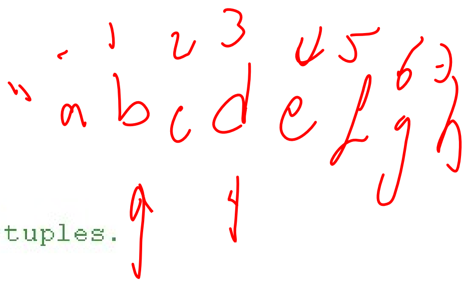
```
the index of ali is:  2
```

# List of tuples

- let us examine a practical example. This program divides a string into a list of tuples. Each tuple has adjacent characters from the string.

**Python that uses list of tuples**

```
value = "abcdefgh"
pairs = []

# Loop over string.
# ... Use step of 2 in range built-in.
# ... Extract pairs of letters into a list of tuples.
for i in range(1, len(value), 2):
    one = value[i - 1]
    two = value[i]
    pairs.append((one, two))

# Display list of tuple pairs.
for pair in pairs:
    print(pair)
```



**Output**

```
('a', 'b')
('c', 'd')
('e', 'f')
('g', 'h')
```

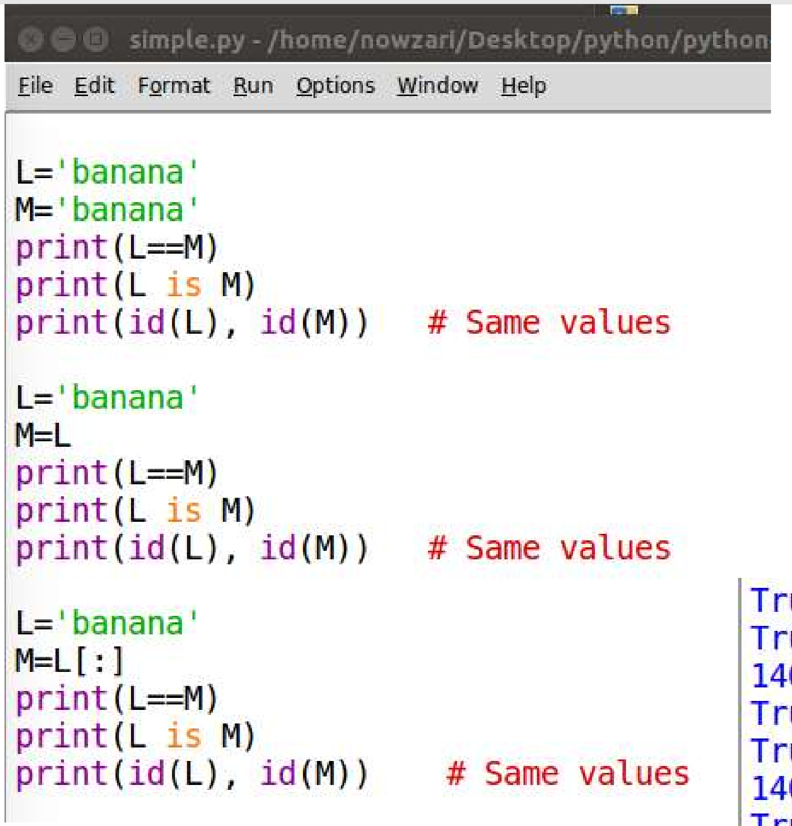
## Shared References

num-simple.py - /home/nowzari/Desktop/python/pyth
File Edit Format Run Options Window Help

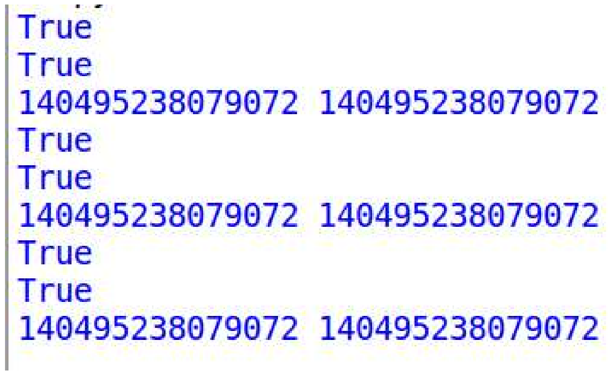
```
X=42
Y=42
print(X==Y)
print(X is Y)

print("id X is", id(X))
print("id Y is", id(Y))

X=40
Y=X
print(X==Y)
print(X is Y)

print("id X is", id(X))
print("id Y is", id(Y))
X=40
Y=5
print(X==Y)
print(X is Y)

print("id X is", id(X))
print("id Y is", id(Y))
```

```
True
True
id X is 10915680
id Y is 10915680
True
True
id X is 10915616
id Y is 10915616
False
False
id X is 10915616
id Y is 10914496
```

# Shared References

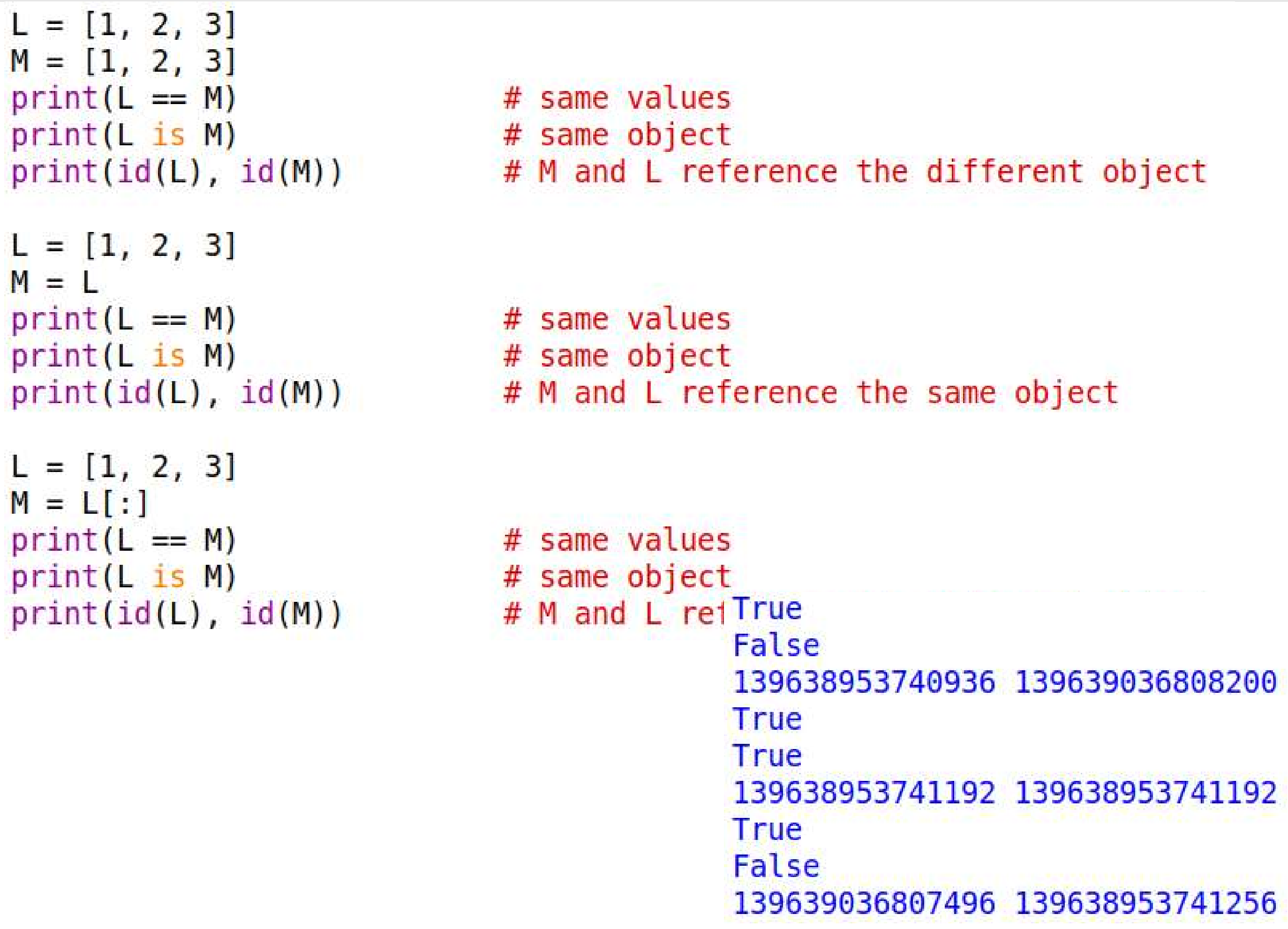

simple.py - /home/nowzari/Desktop/python/python
File Edit Format Run Options Window Help

```
L='banana'
M='banana'
print(L==M)
print(L is M)
print(id(L), id(M))   # Same values

L='banana'
M=L
print(L==M)
print(L is M)
print(id(L), id(M))   # Same values

L='banana'
M=L[:]
print(L==M)
print(L is M)
print(id(L), id(M))    # Same values
```



```
True
True
140495238079072 140495238079072
True
True
140495238079072 140495238079072
True
True
140495238079072 140495238079072
```

## Shared References



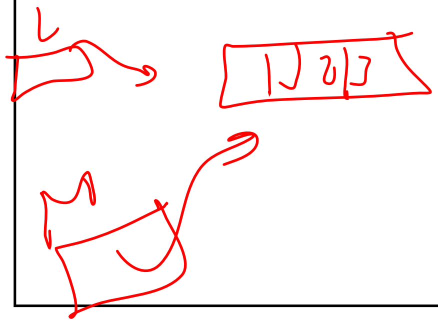
```
L = [1, 2, 3]
M = [1, 2, 3]
print(L == M)           # same values
print(L is M)           # same object
print(id(L), id(M))     # M and L reference the different object

L = [1, 2, 3]
M = L
print(L == M)           # same values
print(L is M)           # same object
print(id(L), id(M))     # M and L reference the same object

L = [1, 2, 3]
M = L[:]
print(L == M)           # same values
print(L is M)           # same object
print(id(L), id(M))     # M and L re
```

```
True
False
139638953740936 139639036808200
True
True
139638953741192 139638953741192
True
False
139639036807496 139638953741256
```

## Share Reference

```
L = (1, 2, 3)
M = (1, 2, 3)
print(L == M)                   # same values
print(L is M)                   # different object
print(id(L), id(M))             # M and L reference the different object

L = (1, 2, 3)
M = L
print(L == M)                   # same values
print(L is M)                   # same object
print(id(L), id(M))             # M and L reference the same object

L = (1, 2, 3)
M = L[:]
print(L == M)                   # same values
print(L is M)                   # same obj
print(id(L), id(M))             # M and L
```

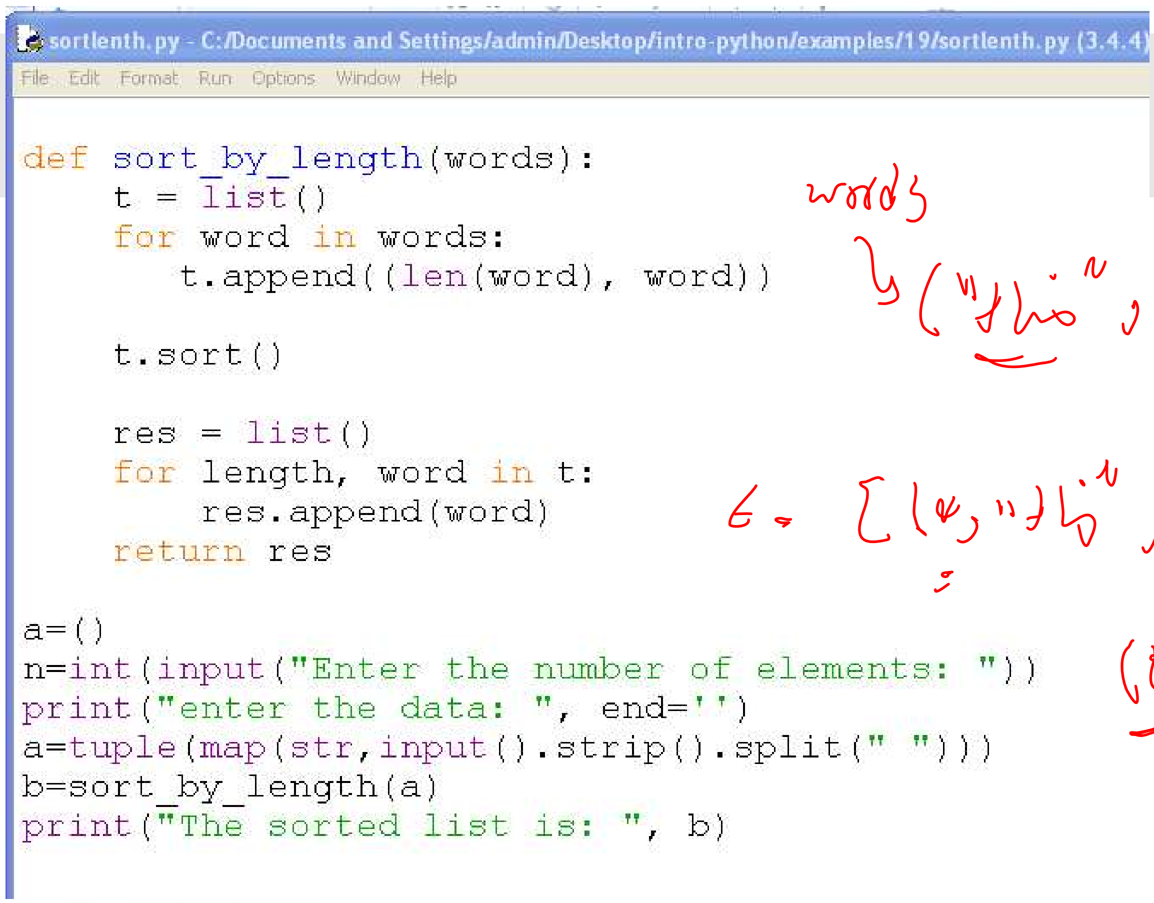
```
True
False
140656957100320 140656991084408
True
True
140657039882208 140657039882208
True
True
140656957026376 140656957026376
```



## Tuple as function’s parameter

- We can send a tuple to a function as a parameter

• .


sortlenth.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/19/sortlenth.py (3.4.4)

```python
def sort_by_length(words):
    t = list()
    for word in words:
       t.append((len(word), word))

    t.sort()

    res = list()
    for length, word in t:
        res.append(word)
    return res

a=()
n=int(input("Enter the number of elements: "))
print("enter the data: ", end='')
a=tuple(map(str,input().strip().split(" ")))
b=sort_by_length(a)
print("The sorted list is: ", b)
```

**Output**

```
Enter the number of elements: 5
enter the data: this is a python course
The sorted list is:  ['a', 'is', 'this', 'course', 'python']
```

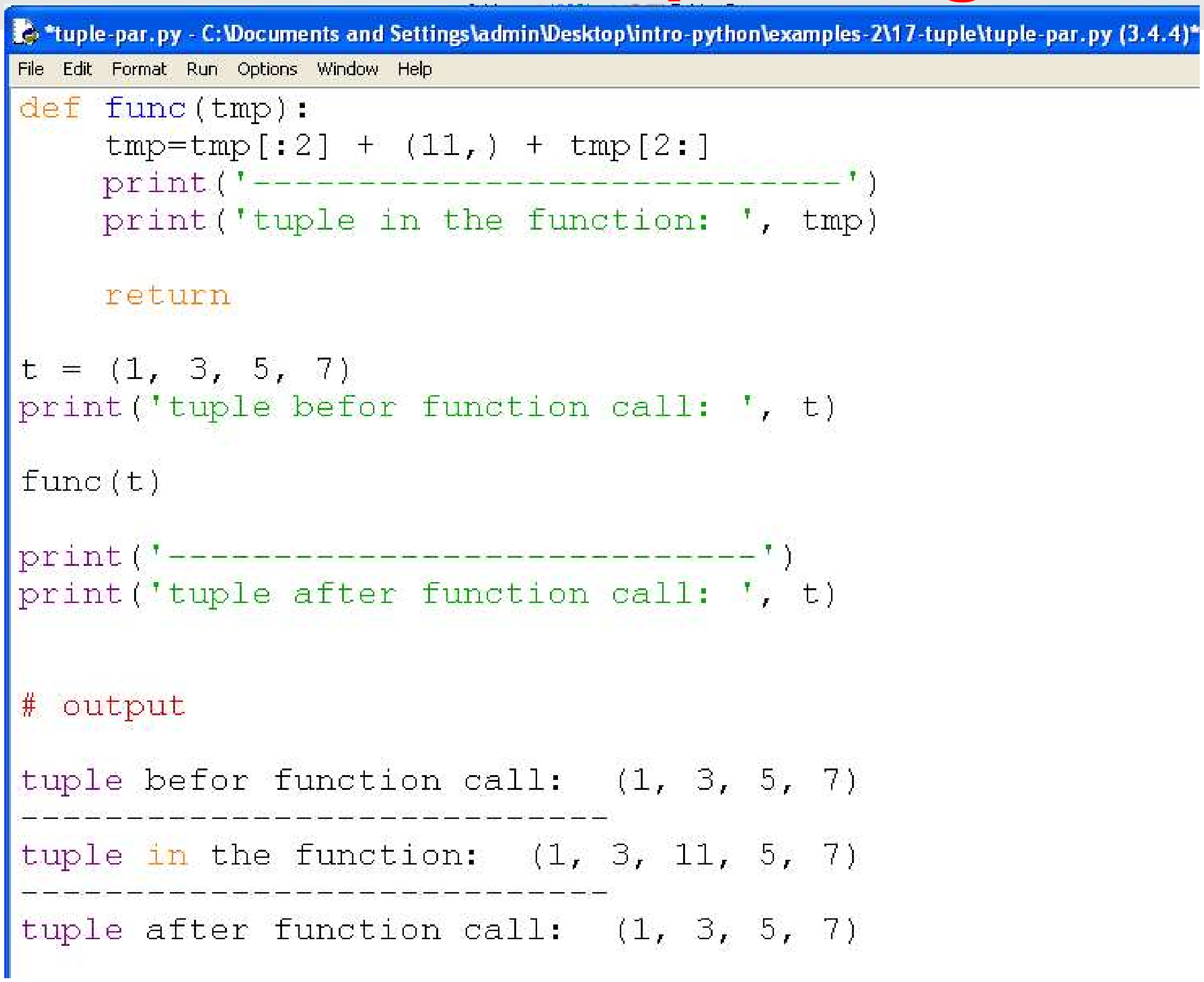
words → ("this", "is", "a", "python", "course")

t = [(4, "this"), (2, "is"), (1, "a"), (6, "python"), (6, "cours")]

## Parameter passing

If you pass immutable arguments like integers, strings or tuples to a function, the passing acts like call-by-value. The object reference is passed to the function parameters. They can't be changed within the function, because they can't be changed at all, i.e. they are immutable.

## Parameter passing

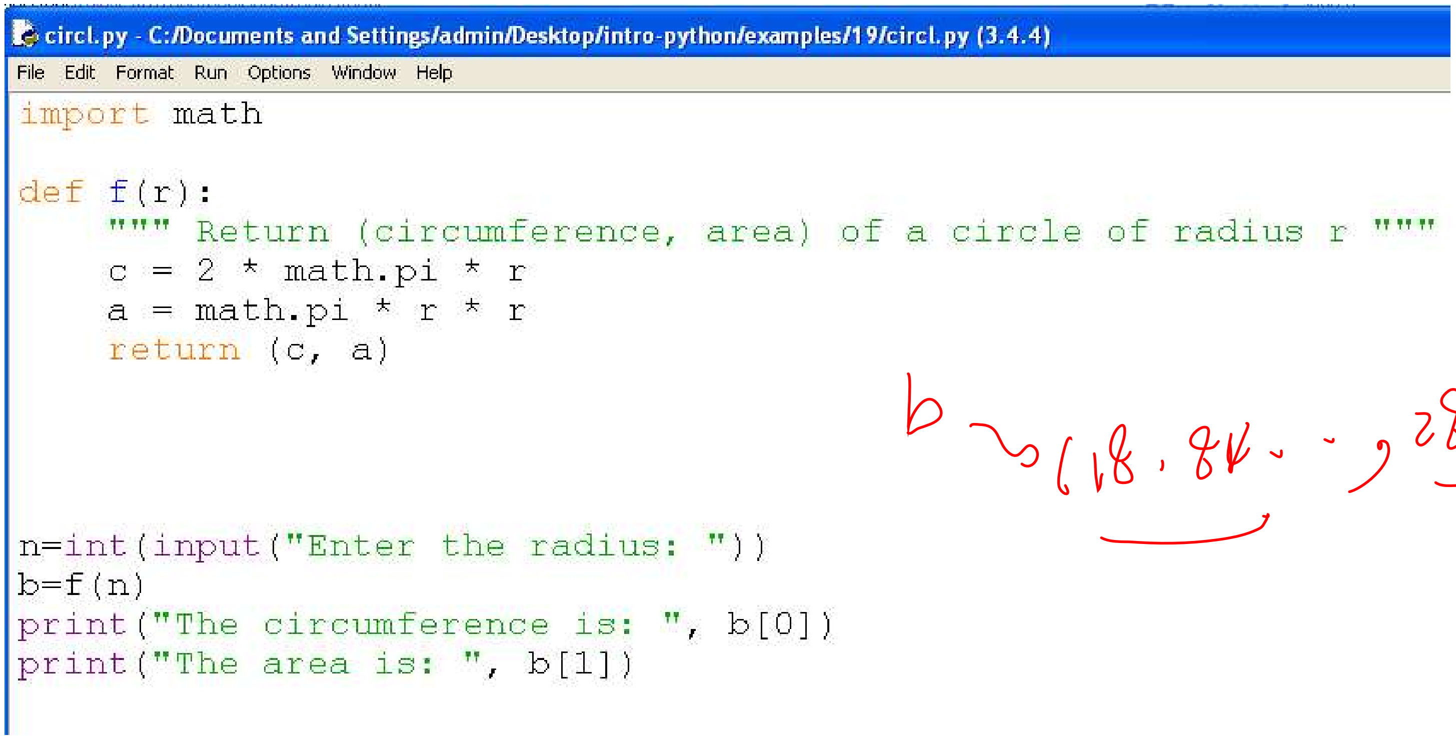

*tuple-par.py - C:\Documents and Settings\admin\Desktop\intro-python\examples-2\17-tuple\tuple-par.py (3.4.4)*
File Edit Format Run Options Window Help

```
def func(tmp):
    tmp=tmp[:2] + (11,) + tmp[2:]
    print('----------------------------')
    print('tuple in the function: ', tmp)

    return

t = (1, 3, 5, 7)
print('tuple befor function call: ', t)

func(t)

print('----------------------------')
print('tuple after function call: ', t)


#  output

tuple befor function call:  (1, 3, 5, 7)
----------------------------
tuple in the function:  (1, 3, 11, 5, 7)
----------------------------
tuple after function call:  (1, 3, 5, 7)
```

## Tuples as return values

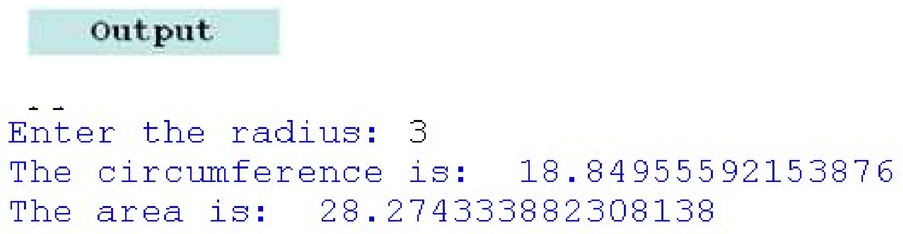
- Functions can always only return a single value, but by making that value a tuple, we can effectively group together as many values as we like, and return them together. This is very useful — we often want to know some batsman's highest and lowest score, or we want to find the mean and the standard deviation, or we want to know the year, the month, and the day, or if we're doing some some ecological modelling we may want to know the number of rabbits and the number of wolves on an island at a given time.
- For example, we could write a function that returns both the area and the circumference of a circle of radius r.

# Tuples as return values


circl.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/19/circl.py (3.4.4)
File Edit Format Run Options Window Help

```python
import math

def f(r):
    """ Return (circumference, area) of a circle of radius r """
    c = 2 * math.pi * r
    a = math.pi * r * r
    return (c, a)



n=int(input("Enter the radius: "))
b=f(n)
print("The circumference is: ", b[0])
print("The area is: ", b[1])
```

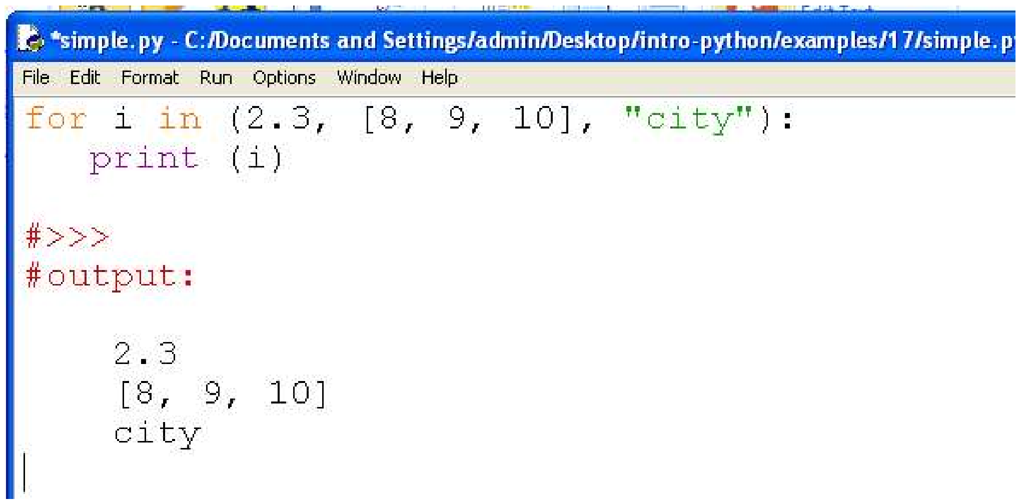
$b \rightsquigarrow (\underline{18.84\ldots}, \underline{28.27\ldots})$



**Output**

```
Enter the radius: 3
The circumference is:  18.84955592153876
The area is:  28.274333882308138
```

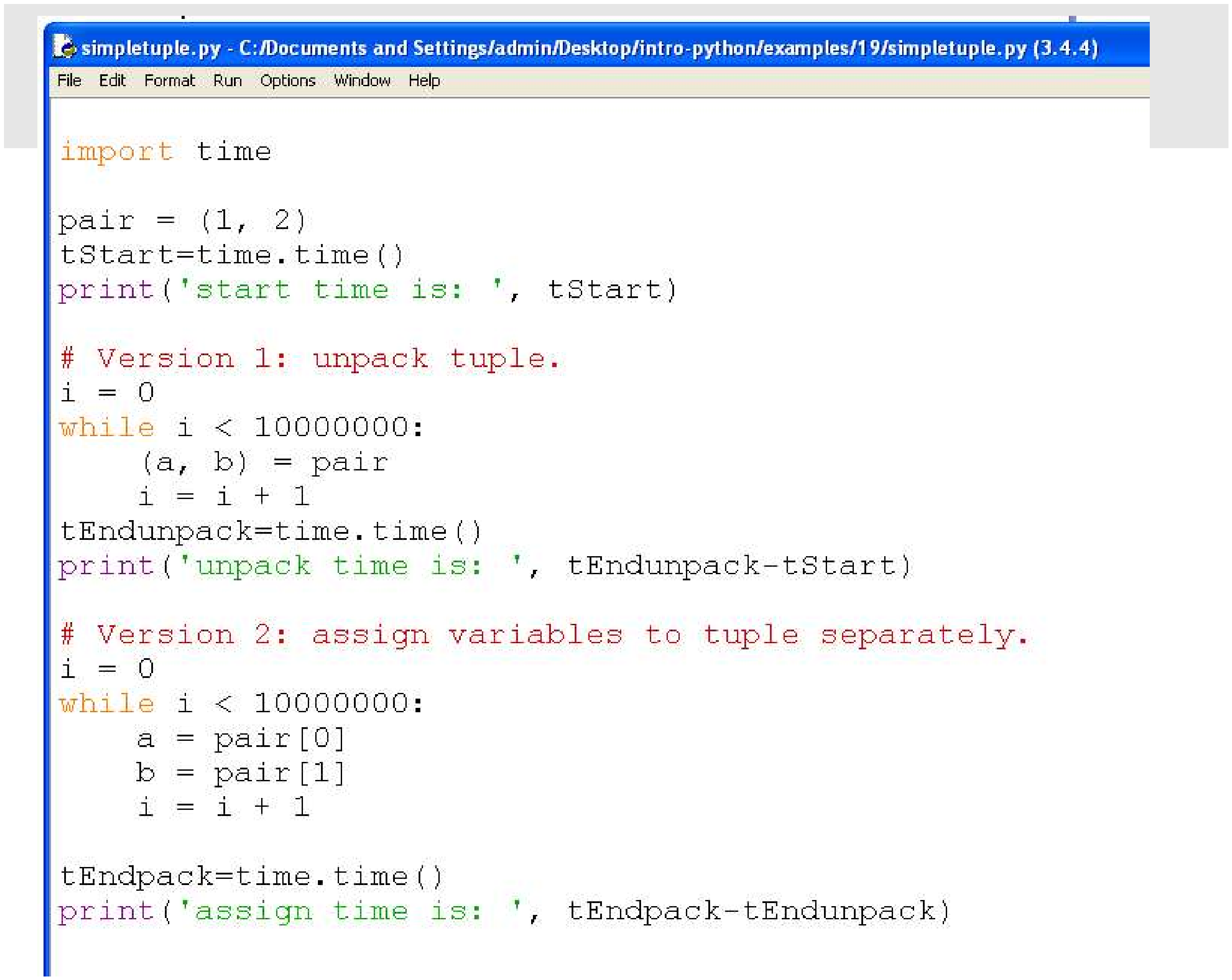
## for loop


*simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/simple.p
File Edit Format Run Options Window Help

```
for i in (2.3, [8, 9, 10], "city"):
    print (i)

#>>>
#output:

    2.3
    [8, 9, 10]
    city
```

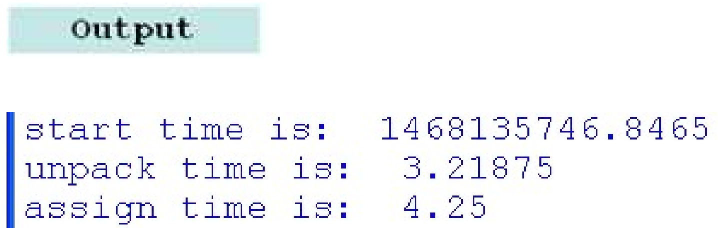
## Performance

- How do we choose between the syntax forms for tuples? In this benchmark we test two ways of assigning variables to values in a tuple. We unpack tuples.


simpletuple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/19/simpletuple.py (3.4.4)
File Edit Format Run Options Window Help

```
import time

pair = (1, 2)
tStart=time.time()
print('start time is: ', tStart)

# Version 1: unpack tuple.
i = 0
while i < 10000000:
    (a, b) = pair
    i = i + 1
tEndunpack=time.time()
print('unpack time is: ', tEndunpack-tStart)

# Version 2: assign variables to tuple separately.
i = 0
while i < 10000000:
    a = pair[0]
    b = pair[1]
    i = i + 1

tEndpack=time.time()
print('assign time is: ', tEndpack-tEndunpack)
```

## Performance



Output

```
start time is:  1468135746.8465
unpack time is:  3.21875
assign time is:  4.25
```

## End