Tuples

Tuesday, March 8, 2022 3:13 PM

1. Tuples

Tuples are an ordered sequence of items similar to lists.

⇒ Also iterable

The general syntax of a tuple is as follows:

$\boxed{(\text{expr }1, \text{ expr }2, \ldots, \text{exprN})}$

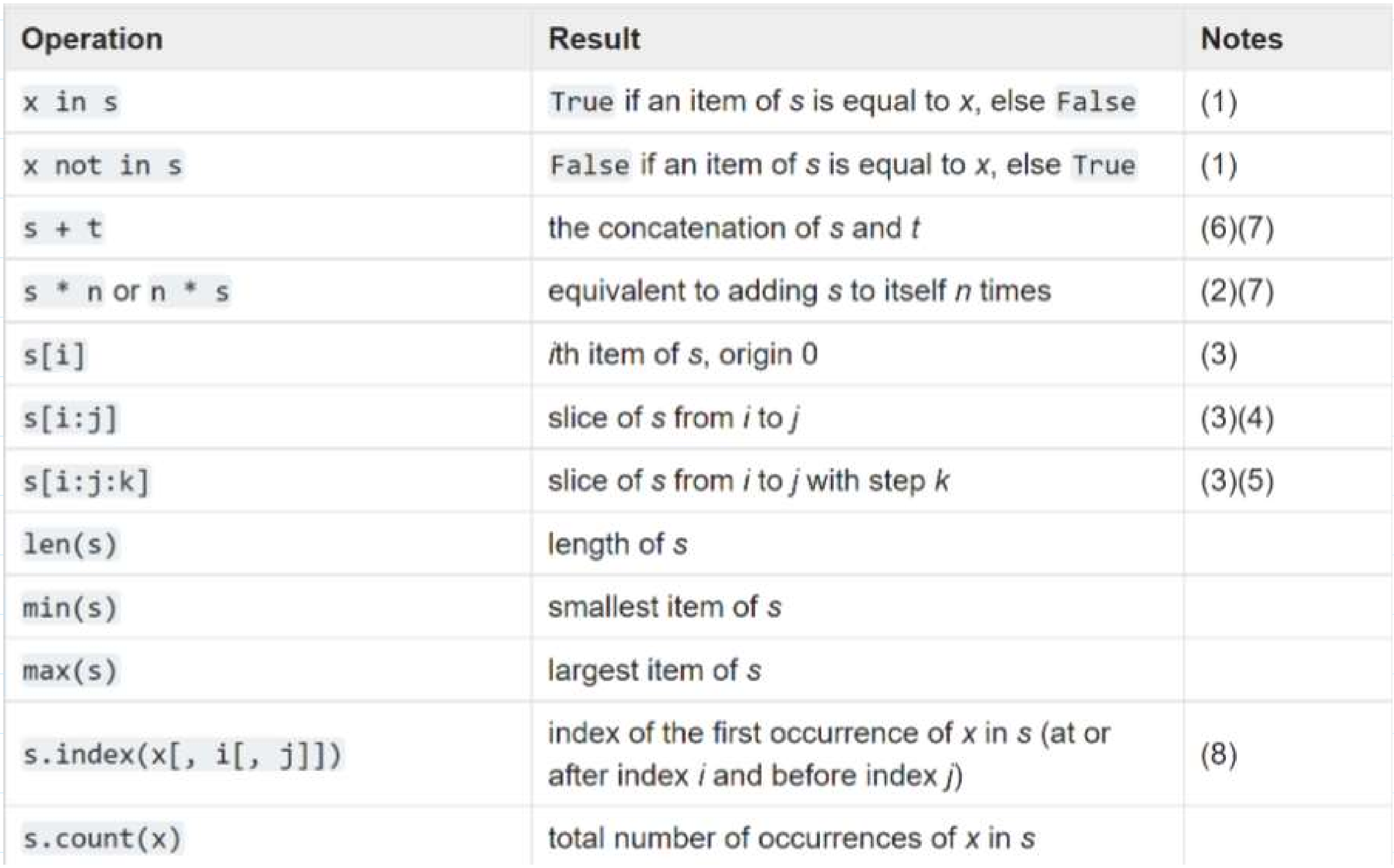
Tuples are represented with parentheses ( ), while lists are represented by [ ]

⇒ To avoid ambiguity with arithmetic operations, we add ","
so a tuple with a single element is written as (expr,)

⊕ $\boxed{\text{Tuples are immutable}}$ ⇒ like strings

⇒ All methods for list can apply, except modifying methods

2. Tuple Operations (like lists)



Operation	Result	Notes
`x in s`	`True` if an item of *s* is equal to *x*, else `False`	(1)
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`	(1)
`s + t`	the concatenation of *s* and *t*	(6)(7)
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times	(2)(7)
`s[i]`	*i*th item of *s*, origin 0	(3)
`s[i:j]`	slice of *s* from *i* to *j*	(3)(4)
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*	(3)(5)
`len(s)`	length of *s*	
`min(s)`	smallest item of *s*	
`max(s)`	largest item of *s*	
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)	(8)
`s.count(x)`	total number of occurrences of *x* in *s*	

NOTE :

+) When assigning variable to tuples , we can include () or not. "," is enough (but () are recommended)

However, when using print () , we must use () or print () will interpret as 3 variables

+) Like strings , you can only concatenate , multiply , or refer to & assign

+) We can convert between tuple and list using tuple () or list ()

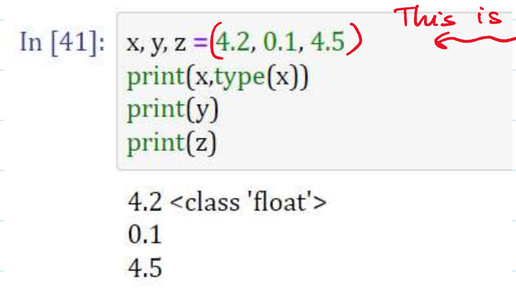
+) A list inside a tuple is still mutable. If we can index, we can still "change the tuple"

Unpacking Tuples

Tuesday, 8 March 2022 22:31

1. Unpacking Tuples

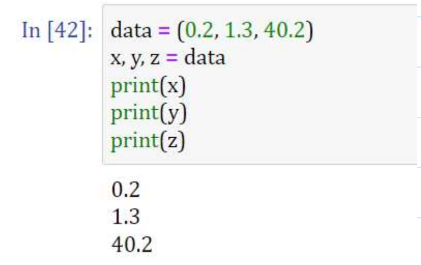
Python has an assignment feature that allows assigning multiple variables at once



```
In [41]: x, y, z =(4.2, 0.1, 4.5)
         print(x,type(x))
         print(y)
         print(z)

4.2 <class 'float'>
0.1
4.5
```

This is a tuple.

≡



```
In [42]: data = (0.2, 1.3, 40.2)
         x, y, z = data
         print(x)
         print(y)
         print(z)

0.2
1.3
40.2
```

We do this by creating a tuple (right side) and "unpack" each values to each variables.

→ We can also unpack a list, given that the number of variables must match len(list) or len(tuple)

→ # Clean Code

2. Tuples as return Values

We can return multiple values from a function using tuples. When doing so, the values are returned in a tuple

→ Unpack and assign to get those values

In [ ]:
```python
import math

def area_circumference(radius):
  """
  (float) -> float, float
  Return the circumference and area of a circle of a specified radius.
  """
  circumference = 2 * math.pi * radius
  area = math.pi * radius * radius

  return circumference, area
```

What is `return`ed?

In [19]:
```python
circumference, area = area_circumference(10)
print(circumference, type(circumference))
print(area, type(area))
```

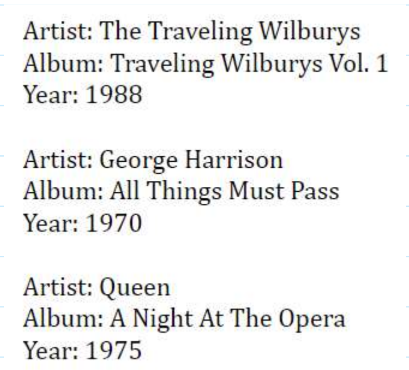
```
62.83185307179586 <class 'float'>
314.1592653589793 <class 'float'>
```

NOTE : We can even assign the unpacking of tuples in for loops :

In [20]:
```python
albums = (("The Beatles", "Sgt. Pepper's Lonely Hearts Club Band", 1967),
("Wintersleep", "Welcome to the Night Sky", 2007),
("The Tragically Hip", "In Between Evolution", 2004),
("Tom Petty", "Wildflowers", 1994),
("The Traveling Wilburys", "Traveling Wilburys Vol. 1", 1988),
("George Harrison", "All Things Must Pass", 1970),
("Queen", "A Night At The Opera", 1975))
```

In [26]:
```python
for artist,album,year in albums:

  print('Artist:', artist)
  print('Album:', album)
  print('Year:', year, '\n')
```

```
Artist: The Beatles
Album: Sgt. Pepper's Lonely Hearts Club Band
Year: 1967

Artist: Wintersleep
Album: Welcome to the Night Sky
Year: 2007

Artist: The Tragically Hip
Album: In Between Evolution
Year: 2004

Artist: Tom Petty
Album: Wildflowers
Year: 1994
```



```
Artist: The Traveling Wilburys
Album: Traveling Wilburys Vol. 1
Year: 1988

Artist: George Harrison
Album: All Things Must Pass
Year: 1970

Artist: Queen
Album: A Night At The Opera
Year: 1975
```

Sets

Tuesday, 8 March 2022 22:51

Sets are $\boxed{\text{unordered}}$ collection of distinct items that does not record element positions → Iterable

→ Sets do not support indexing, slicing, etc

$$\boxed{\{\ exp1,\ exp2,\ \ldots,\ expN\ \}}$$

Their main purpose is to hold items.
There are no duplicates in sets.

→ Even if we add duplicated items, that is not additionally added.

Set Operations :

Operation	Equivalent	Result
len(s)	N/A	number of elements in set s (cardinality)
x in s	N/A	test x for membership in s
x not in s	N/A	test x for non-membership in s
s.issubset(t)	s <= t	test whether every element in s is in t
s.issuperset(t)	s >= t	test whether every element in t is in s
s.union(t)	s | t	new set with elements from both s and t
s.intersection(t)	s & t	new set with elements common to s and t
s.difference(t)	s - t	new set with elements in s but not in t
s.symmetric_difference(t)	s ^ t	new set with elements in either s or t but not both
s.copy()	N/A	new set with a copy of s

Sets are mutable

1. Membership Use in operator

→ Test whether an element is in the set

2. Union The Union of 2 or more sets is the Set

of all items that appear across all sets

No duplicates ⇒ remove all duplicates

+) europe . union (north_america)
+) north _ america | europe

3. Intersection Set of all items that are in each set.

+) north_america. intersection (europe)
+) europe & north_america

Dictionaries

Thursday, March 10, 2022 3:09 PM

Dictionaries are unordered data structure similar to sets
Dictionaries contain references to objects as key : value pair
Dictionaries are mutable.

$\boxed{\{ \text{key1 : val1 , key2 : val 2 , ... , keyN : valN} \}}$

keys : Must be immutable (no lists or sets)
values : Anything (even dictionaries)

* Operations

Operation	Description	Example code
my_dict[key]	Indexing operation – retrieves the value associated with key.	john_grade = my_dict['John']
my_dict[key] = value	Adds an entry if the entry does not exist, else modifies the existing entry.	my_dict['John'] = 'B+'
del my_dict[key]	Deletes the key:value from a dict.	del my_dict['John']
key in my_dict	Tests for existence of key in my_dict	if 'John' in my_dict: ...

→ Only check for keys

Note : Creating empty {} creates a dictionary. If we want to create an empty set, we need to use

```
In [12]: data = set()
         print(type(data))

<class 'set'>
```

• Methods   Dictionaries are mutable ⇒ Only call method.

+) .keys() → Return set-like object containing all keys
+) .values() → Return list-like object containing all values.
+) .items() → Return list-like object containing tuples of key-value pairs
+) .clear() → Remove all elements
+) .get() → Return value of key entry (None / second argument if key doesn't exist)
+) .update() → Merges dict1 with dict2. Like sets, no duplicates
→ value in dict1 overwritten by value in dict2.
+) .pop() → Removes and return value corresponding to specified key to assigned variable.
None / second argument if key do not exist.

Write a program to rename the `'city'` key to be called `'location'` in the following dictionary.

```
In [29]: dict5 = {
          "name": "Seb",
          "age": 36,
          "salary": 8000,
          "city": "Toronto"
         }

         # Write your code here
         dict5['location'] = dict5.pop('city')
         print(dict5)

         #Since dict has no orders, this will always works

         {'name': 'Seb', 'age': 36, 'salary': 8000, 'location': 'Toronto'}
```

- Iterating over dictionaries:

```
friends = {"Bob": 32, "Jane: 42}
```

- Keys : Default will loop over keys

```
>>> for key in friends:
        print(key)
    "Bob"
    "Jane"
```

- Values : Use .values() to get list-like object of values

```
>>> for value in friends.value():
        print(value)
    32
    42
```

- Keys and Values : Use .items() to get list like object of tuples.

```
>>> for item in friends.items():
        print(item)
    ("Bob", 32)
    ("Jane", 42)
```

Since items in list-like .items() are tuples, we can unpack items and values into different variables.

```
>>> for name, age in friends.item():
        print(name, age)
    "Bob"  32
    "Jane"  46
```

Note: If we loop over the dictionary, pay attention to the type of the item being used in each loop.

Example

```
students = {"Michael": {"Final": 100}, "Scott": {"Final": 98} }
for name in students:
    print(name["Final"])  ✗ !!!
```

⇒ Error since name is string.
⇒ Need to use students[name]["Final"]
→ 100