Introduction to PCEP Python

Class 9: Tuples and Dictionaries

Sequence Types

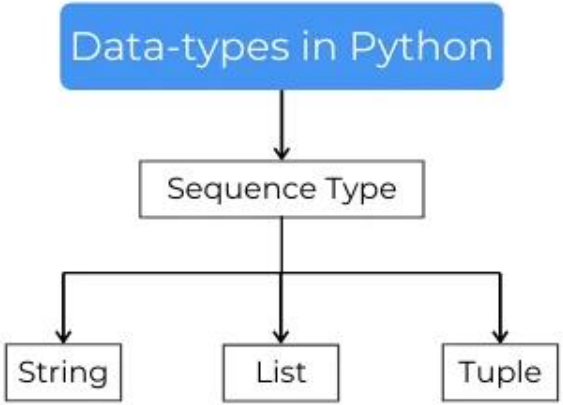
- A **sequence type** is a type of data in Python which is able to store more than one value (or less than one, as a sequence may be empty), and these values can be sequentially (hence the name) browsed, element by element.

- As the for loop is a tool especially designed to iterate through sequences, we can express the definition as: a sequence is data which can be scanned by the for loop.

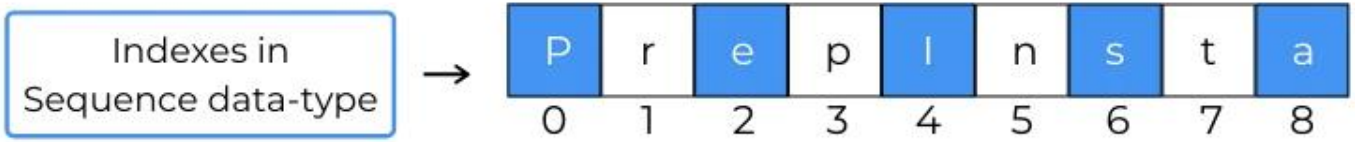
- A list is an example of a Python sequence.

Sequence Types

Sequence data-types in Python



- List data-type is an ordered sequence of collection of elements. It is mutable data-type.
- The string is a byte representation of uni-code characters. Elements of strings are accessed using square brackets.
- Tuples are data-type same as a list, but tuples are immutable means they are not modified.



PrepInsta

Mutability

- **Mutability** is a property of any of Python's data that describes its readiness to be freely changed during program execution. There are two kinds of Python data: mutable and immutable.
    - Mutable data can be freely updated at any time.
    - Immutable data cannot be modified in this way.

- For example, a mutable list would be one that you could append new elements to. You could not add items to an immutable list, and would have to create a new list with the new items you want to add.

Mutable vs Immutable Objects

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Tuples

- A **tuple** is an immutable sequence type. It can behave like a list but cannot have operations such as adding and removing elements.
- Each tuple element may be of a different type (floating-point, integer, or any other not-as-yet-introduced kind of data).
- It is possible to create an empty tuple - parentheses are required: *empty_tuple = ()*
- To create a tuple with elements in it, you can do it either of these two ways:

```
tuple_1 = (1, 2, 4, 8)
tuple_2 = 1., .5, .25, .125
```

Tuples

PYnative.com

Tuples in Python

T = (20, 'Jessa', 35.75, [30, 60, 90])

T[0] T[1] T[2] T[3]

- ✓ **Ordered**: Maintain the order of the data insertion.
- ✓ **Unchangeable**: Tuples are immutable and we can't modify items.
- ✓ **Heterogeneous**: Tuples can contains data of types
- ✓ **Contains duplicate**: Allows duplicates data

Tuples vs. Lists

- The main difference between lists and tuples is that lists are mutable, while tuples are immutable.
    - Once a list is created, its contents can be changed.
        - Lists can be modified by adding, removing, or changing elements.

    - Once a tuple is created, its contents cannot be changed.
        - If you need to modify a tuple, you must create a new tuple with the desired changes.

Tuples vs. Lists

- Another difference is that lists are typically used for collections of similar items that may need to be modified over time, while tuples are typically used for collections of dissimilar items that should not be modified.

- For example, if you were creating a sequence of student names, you might use a list, since you may need to add or remove students over time.

- If you were creating a tuple to represent a point in space, you might use a tuple, since the coordinates of a point should not change.

Tuples vs. Lists

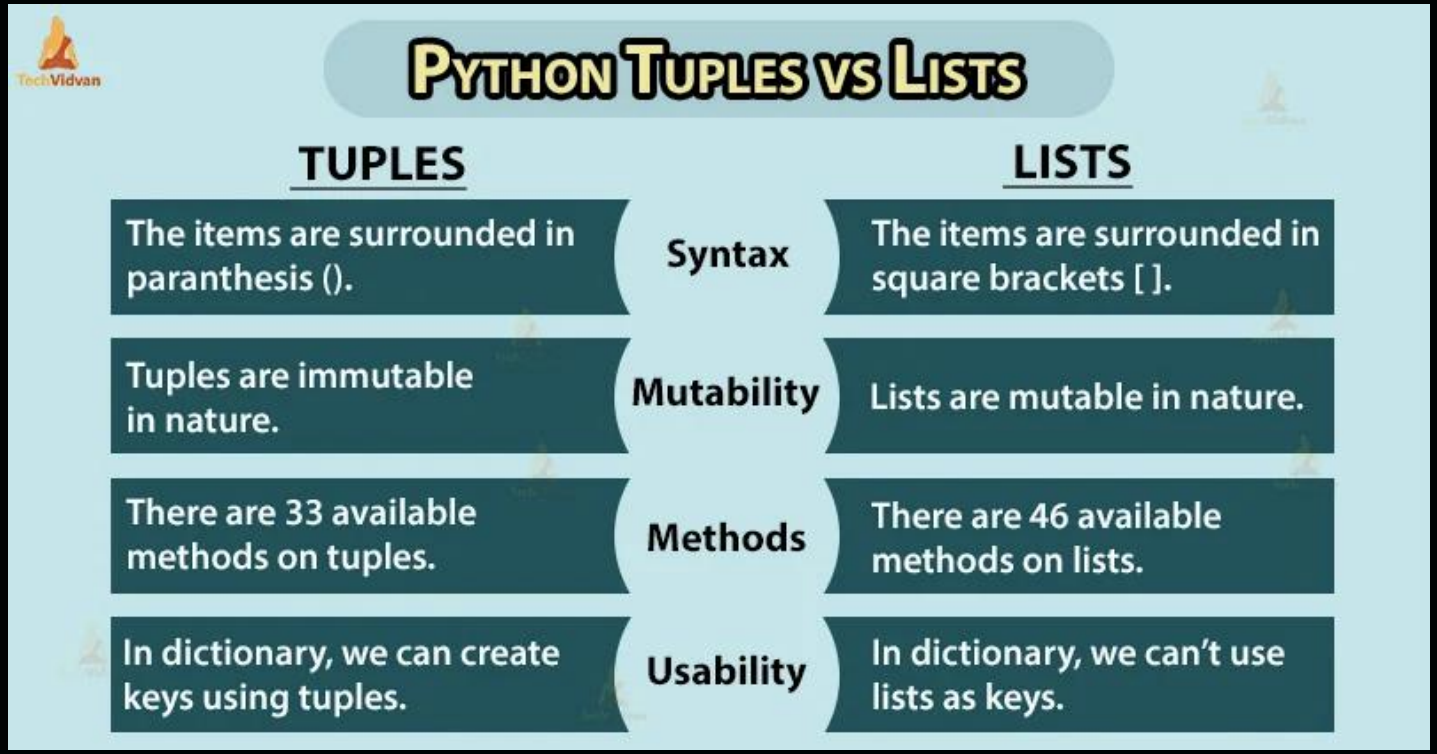
```
# creating a list
my_list = [1, 2, 3, "four", "five"]

# accessing elements in a list
my_list[0]  # returns 1
my_list[3]  # returns "four"

# creating a tuple
my_tuple = (10, 20, 30, "forty", "fifty")

# accessing elements in a tuple
my_tuple[1]  # returns 20
my_tuple[4]  # returns "fifty"
```

Tuples vs. Lists



Accessing Elements of a Tuple

```
1  my_tuple = (1, 10, 100, 1000)
2
3  print(my_tuple[0])
4  print(my_tuple[-1])
5  print(my_tuple[1:])
6  print(my_tuple[:-2])
7
8  for elem in my_tuple:
9      print(elem)
10
```

Console >_

```
1
1000
(10, 100, 1000)
(1, 10)
1
10
100
1000
```

Other Uses of Tuples

- the *len()* function accepts tuples, and returns the number of elements contained inside;
- the + operator can join tuples together (we've shown you this already)
- the * operator can multiply tuples, just like lists;
- the *in* and *not in* operators work in the same way as in lists.

Tuple Functions

Function	Description
cmp(tuple1,tuple2)	Compares elements of two different tuples.
len(tuple)	Returns the length of the tuple
max(tuple)	Returns the element with max value from the tuple.
min(tuple)	Returns the element with min value from the tuple.
tuple(seq)	Returns a tuple, by converting a list to a tuple. Takes a list in the parameter.

SCALER Topics

Dictionaries

- A dictionary is a collection of key-value pairs, where each key is associated with a value. Dictionaries are an important data structure in Python and are commonly used for tasks such as storing configuration settings, mapping between two sets of data, and building complex data structures.

- To create a dictionary in Python, you use curly braces ({}) and separate each key-value pair with a colon (:). Here's an example:

```
my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```

- the keys are 'key1', 'key2', and 'key3', and the corresponding values are 'value1', 'value2', and 'value3'.

Dictionaries

- A dictionary works like a translator, or an English to Spanish dictionary, for example.
  - In a translator, if you have an English word (e.g., cat) and want to look up its Spanish equivalent, you browse the dictionary in order to find the word (you may use different techniques to do that - it doesn't matter) and eventually you get it. Next, you check the Spanish counterpart and it would be the word "gato".

Dictionaries

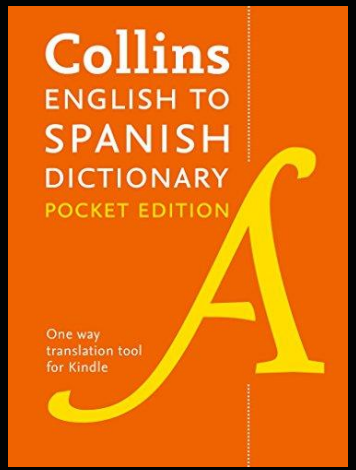
- Each key in a dictionary must be unique - it's not possible to have more than one key of the same value; a key may be any immutable type of object: it can be a number (integer or float), or even a string, but not a list.

- A dictionary is not a list - a list contains a set of numbered values, while a dictionary holds pairs of values.

Dictionaries

- The *len()* function works for dictionaries, too - it returns the numbers of key-value elements in the dictionary.

- A dictionary is a one-way tool - if you have an English-Spanish dictionary, you can look for Spanish equivalents of English terms, but not vice versa.



Dictionaries

Dictionary in Python

PYnative.com

Unordered collections of unique values stored in (Key-Value) pairs.

```
d = {'a': 10, 'b': 20, 'c': 30}
```

d['a'] d['b'] d['c']

- ✓ **Unordered**: The items in dict are stored without any index value
- ✓ **Unique**: Keys in dictionaries should be Unique
- ✓ **Mutable**: We can add/Modify/Remove key-value after the creation

Creating a Dictionary in Python

```
In [1]: # Creating a simple dictionary
        dict1={101:'ram',
               102:'sham',
               103:'ravi'}

        # printing the whole dictionary
        print(dict1)

        # accessing a value using its key
        dict1[103]
```

```
{101: 'ram', 102: 'sham', 103: 'ravi'}

Out[1]: 'ravi'
```

Accessing Elements in a Dictionary: Square Brackets

- There are two ways to access elements in a dictionary:
  - You can access the value associated with a key in a dictionary using square brackets:

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

# Accessing value associated with key 'a'
print(my_dict['a']) # Output: 1
```

Accessing Elements in a Dictionary: get() method

- If the key is not in the dictionary, a KeyError will be raised. To avoid this, you can use the get() method:
  - The get() method returns the value associated with the key if it exists, and a default value (in this case, 'Not found') if the key is not present.

```
Accessing value using get()
print(my_dict.get('d', 'Not found'))
# Output: Not found
```

Accessing Elements in a Dictionary

In [3]:
```
#Accessing elements using square brackets
d1 = {1:"welcome", 2:"to", 3:"python", 4:"tutorial"}
print(d1[1])

#Accessing element using get() method
d1 = {1:"welcome", 2:"to", 3:"python", 4:"tutorial"}
print(d1.get(1))
```

```
welcome
welcome
```

Modifying Elements in a Dictionary: Adding a Key

- Adding a new key-value pair to a dictionary is as simple as changing a value - you only have to assign a value to a new, previously non-existent key.

```
1  dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}
2
3  dictionary['cat'] = 'minou'
4  print(dictionary)
5
```

Console >_

```
{'cat': 'minou', 'dog': 'chien', 'horse': 'cheval'}
```

Modifying Elements in a Dictionary: Adding a Key Using update()

- You can also insert an item to a dictionary by using the update() method.

```
1  dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval
2
3  dictionary.update({"duck": "canard"})
4  print(dictionary)
5
6
7
8
9
10
```

Console >_

```
{'cat': 'chat', 'dog': 'chien', 'horse': 'cheval', 'duck': 'canard'}
```

Modifying Elements in a Dictionary: Removing a Key

- Removing a key in Python is done with the del instruction. Removing a key will always cause the removal of the associated value. Values cannot exist without their keys.
- Removing a non-existent key will cause an error.

```
1  dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval
2
3  del dictionary['dog']
4  print(dictionary)
5
6
7
8
9
10
```

Console >_

```
{'cat': 'chat', 'horse': 'cheval'}
```

Looping Through a Dictionary: keys() Method

- In Python, you can loop over the keys, values, or items in a dictionary using a for loop or a comprehension
  - You can loop over the keys in a dictionary using the keys() method:

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

for key in my_dict.keys():
    print(key)

#Output:
a
b
c
```

Looping Through a Dictionary: values() method

- You can loop over the values in a dictionary using the values() method:

```
dictionary = {"cat": "chat", "dog": "chien",
"horse": "cheval"}

for french in dictionary.values():
    print(french)


    #Output:
    chat
    chien
    cheval
```

Looping Through a Dictionary: items() method

- You can loop over the key-value pairs in a dictionary using the items() method:

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

for key, value in my_dict.items():
    print(key, value)


#Output:
a 1
b 2
c 3
```

Sorting a Dictionary in Python

Python - Sort a Dictionary

- We can sort a Python DIctionary based on the Key or by Value
- For Python versions 3.6+, we can use a lambda, along with sorted()
- Otherwise, we can use operator.itemgetter(idx) on a sorted list

```
>>> my_dict = {1: 10, 4: 12, 2: 100}
>>> dict(sorted(my_dict.items(), key=lambda item: item[0]))
        {1: 10, 2: 100, 4: 12}
```

Challenge Questions

55) Assuming that tuple is a correctly created tuple, the fact that tuples are immutable means that the following instruction:

```
tuple[1] = tuple[1] + tuple[0]
```

- is illegal
- is fully correct
- can be executed if and only if the tuple contains at least two elements
- may be illegal if the tuple contains strings

55) Assuming that tuple is a correctly created tuple, the fact that tuples are immutable means that the following instruction:

```
tuple[1] = tuple[1] + tuple[0]
```

- is illegal
- is fully correct
- can be executed if and only if the tuple contains at least two elements
- may be illegal if the tuple contains strings

56) What is the output of the following snippet?

```
list = [‘Mary’, ‘had’, ‘a’, ‘little’, ‘lamb’]
def list(L):
del L[3]
L[3] = ‘ram’
print(list(list))
```

- [‘Mary’, ‘had’, ‘a’, ‘ram’]
- [‘Mary’, ‘had’, ‘a’, ‘little’, ‘lamb’]
- the snippet is erroneous
- [‘Mary’, ‘had’, ‘a’, ‘lamb’]

56) What is the output of the following snippet?

```
list = [‘Mary’, ‘had’, ‘a’, ‘little’, ‘lamb’]
def list(L):
del L[3]
L[3] = ‘ram’
print(list(list))
```

- [‘Mary’, ‘had’, ‘a’, ‘ram’]
- [‘Mary’, ‘had’, ‘a’, ‘little’, ‘lamb’]
- the snippet is erroneous
- [‘Mary’, ‘had’, ‘a’, ‘lamb’]

59) What is the output of the following snippet?

```
dct = { 'one':'two', 'three':'one', 'two':'three' }
v = dct['one']
for k in range(len(dct)):
v = dct[v]
print(v)
```

- two
- three
- ('one', 'two', 'three')
- one

59) What is the output of the following snippet?

```
dct = { 'one':'two', 'three':'one', 'two':'three' }
v = dct['one']
for k in range(len(dct)):
v = dct[v]
print(v)
```

- two
- three
- ('one', 'two', 'three')
- one

60) What is the output of the following snippet?

```
tup = (1, 2, 4, 8)
tup = tup[1:-1]
tup = tup[0]
print(tup)
```

- the snippet is erroneous
- (2)
- (2,)
- 2

60) What is the output of the following snippet?

```
tup = (1, 2, 4, 8)
tup = tup[1:-1]
tup = tup[0]
print(tup)
```

- the snippet is erroneous
- (2)
- (2,)
- 2

Review Questions

Exercise 2

What is the output of the following snippet?

```
def fun(a):
    if a > 30:
        return 3
    else:
        return a + fun(a + 3)


print(fun(25))
```

Check

Exercise 2

What is the output of the following snippet?

```
def fun(a):
    if a > 30:
        return 3
    else:
        return a + fun(a + 3)


print(fun(25))
```

Check

```
56
```

Exercise 1

What will happen when you attempt to run the following snippet and why?

```
def factorial(n):
    return n * factorial(n - 1)


print(factorial(4))
```

Check

Exercise 1

What will happen when you attempt to run the following snippet and why?

```
def factorial(n):
    return n * factorial(n - 1)


print(factorial(4))
```

Check

The factorial function has no termination condition (no base case) so Python will raise an exception (`RecursionError: maximum recursion depth exceeded`)