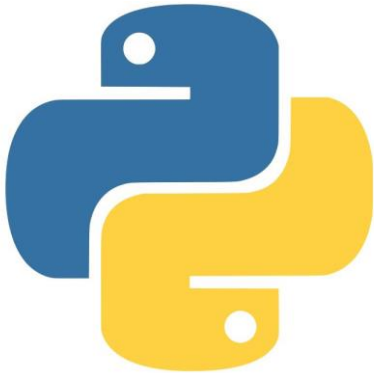
python



PROGRAMMING ESSENTIALS WEBINAR

DAY 6

Contents

tuples

Nature of Tuples

- A tuple is a collection of objects which are ordered and *immutable*.
- Differences between tuples and lists are:
  - Tuples' items cannot be changed (immutability), unlike lists.
  - Tuples use ( ) symbols where as lists use [ ].
- When to use?
  - If data should not be changed in any part of the code.

Tuples | Instantiation

```
tup = ()
tup = tuple()
tupleOne = (item1, item2, itemN)
tupleTwo =(item1, )
tupleThr = item1, item2, itemN
tupleFou = item1,
```

Tuples | Indexing and Looping

```
daysOfWeek = ('Sunday', 'Monday',
'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday')

print(daysOfWeek[0])

#Sunday

for day in daysOfWeek:
    print(day)
```

Tuples | Deletion

```
daysOfWeek = ('Sunday', 'Monday',
'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday')

del daysOfWeek
```

Tuples | Concatenating Tuples

```
daysOfWeek = ('Sunday', 'Monday',
'Tuesday', 'Wednesday', 'Thursday',
'Friday')

daysOfWeek = daysOfWeek + ('Saturday',)
```

Tuples | Conversion

Sequence conversion is possible between lists and tuples.

`list()` – Built-in function that returns a list

`tuple()` – Built-in function that returns a tuple()

`type()` – Built-in function that returns an object's type

Tuples | Multiple Variable Assignment

In Python, multiple variables can be instantiated with initial values in one line, by using tuples.

```
a, b, some_list = 1, 2, [3,4,5]
```

Tuples | Additional Built-in functions

The max() built-in function returns the largest item based on the list or tuple passed into its parameter. Alternatively, the min() built-in function returns the item with the smallest value based on the sequenced passed into its parameter.

```
a, b = max((1, 2)),min((5,1,3))
c = sum((a, b))
print(a)
print(b)
print(c)
```

dictionaries

Nature of Dictionaries

- Hold key-value pairs or *items*.
- Indices are known as *keys. Keys* can be strings or integers.
- Also known as associative arrays, hash tables and hashes.
- Bound by curly brackets { }.
- Each item is expressed as key : value and separated by a comma.
- Mutable

item

- Keys should be unique in a dictionary – where as values may not be.

Dictionaries | Instantiation

```
dictionary = {}
dictionary = dict()
#assigning values
dictionary = {“name”: “Guido”,
“designation”: [“Python BDFL”, “Developer”,
“Programmer at MS”]}
#referencing values
dictionary[key_name]
```

Dictionaries | Indexing

```
students = {'Jo' : 77, 'JoJo': 74, 'Joe': 93}
print(f'JoJo had a grade of {students['JoJo']}.')
# JoJo had a grade of 74
```

Dictionaries | Mutability

```
students = {'Jo' : 77, 'JoJo': 74, 'Joe': 93}
print(f'JoJo had a grade of {students['JoJo']}.')
# JoJo had a grade of 74
students['JoJo'] = 75
print(f'JoJo had a grade of {students['JoJo']}.')
# JoJo had a grade of 75
```

Dictionaries | Len

```
students = {'Jo' : 77, 'JoJo': 75, 'Joe': 93}
student_count = len(students)
print('We have {} students!'.format(student_count))
# We have 3 students!
```

Dictionaries | Removing Items

```
students = {'Jo' : 77, 'JoJo': 75, 'Joe': 93}
del students['JoJo']
print(students)
# {'Jo': 77, 'Joe': 93}
```

Dictionaries | Adding Items

```
students = {'Jo' : 77, 'Joe': 93}
students['JoJoe'] = 99
print(students)
# {'Jo': 77, 'Joe': 93, 'JoJoe': 99 }
```

Looping | Dictionary Keys

```
students = {
    'Jo' : 77,
    'JoJoe': [ 99, 95 ],
    'Joe': 93
}
for student in students.keys():
    print(student)
# Jo
# Joe
# JoJoe
```

Looping | Dictionary Values

```
students = {
    'Jo' : 77,
    'JoJoe': [ 99, 95 ],
    'Joe': 93
}
for student in students.values():
    print(student)
```

Looping | Dictionary Items

```
students = {
    'Jo' : 77,
    'JoJoe': [ 99, 95 ],
    'Joe': 93
}
for student, grades in students.items():
    print(f"{student} got the grade {grades}")
```

sets

Nature of Sets

- Sets are a collection of items which are unordered and unindexed.
- These are iterable and mutable.
- Contains *no duplicate values*.
- Bound by the { } symbols.

Sets | Instantiation and adding values

```
new_set = set()
some_set = {1,2,3}
new_set.add(1)
new_set.add(2)
new_set.add(2)
print(new_set)
# {1,2}
```

```
new_set.update([1,2])
print(new_set)
# {1,2,[1,2]}
new_set.add([3,4])
# TypeError
new_set.update(5)
# TypeError
```

Sets | add vs update

- Quick note: use .add() – if you're adding constants (individual elements), where as .update() – if you're adding iterables (i.e. lists, tuples, sets, dictionaries)
- However, both add() and update() can cater to strings, however strings will be treated differently by both functions.

```
a = set()
a.add('hello')
print(a)
# {'hello'}
a.update('hello')
print(a)
# {'e', 'h', 'hello',
'o', 'l'}
```

Built-in Methods | add

```
new_set = set()
some_list = [1,5,1,1,2,3,3,1,4,5]
for i in some_list:
    new_set.add(i)
print(new_set)
# {1,2,3,4,5}
```

Built-in Methods | remove

```
new_set = {1,2,3}
some_list = [1,5,1,1,2,3,3,1,4,5]
for i in some_list:
    if i in new_set
        new_set.remove(i)
print(new_set)
# set()
```

Built-in Methods | discard

```
new_set = {1,2,3}
some_list = [1,5,1,1,2,3,3,1,4,5]
for i in some_list:
    if i in new_set
        new_set.discard(i)
print(new_set)
# set()
```

Sets | remove vs discard

- Both set functions remove an item from the set.
- However if an item doesn't exist in the set – remove raises a KeyError, where as discard does not – and just does nothing.

```
example = {1,2,3}
example.discard(4)
#nothing happens
example.remove(4)
# KeyError
```

Built-in Methods | clear

```
new_set = {1,2,3,4}
print(new_set)
# {1,2,3,4}
new_set.clear()
print(new_set)
# set()
```

Built-in Methods | Union (logical sum)

```
even_set = set()
odd_set = set()
for x in range(11):
    if x % 2 == 0:
        even_set.add(x)
    else:
        odd_set.add(x)
all_nums = even_set.union(odd_set)
print(all_nums)
# prints {0,1,2,3,4,5,6,7,8,9,10}
```

Built-in Methods | Intersection (logical product)

```
even_set = {0,2,4,6,8,10}
some_set = {0,3,6,9}
all_nums = even_set.intersection(some_set)
print(all_nums)
# {0,6}
```

Built-in Methods | difference

```
even_set = {0,2,4,6,8,10}
some_set = {0,3,6,9}
all_nums = even_set.difference(some_set)
print(all_nums)
# {8,2,10,4}
other = some_set.difference(even_set)
# {3,9}
```