**Tuple**
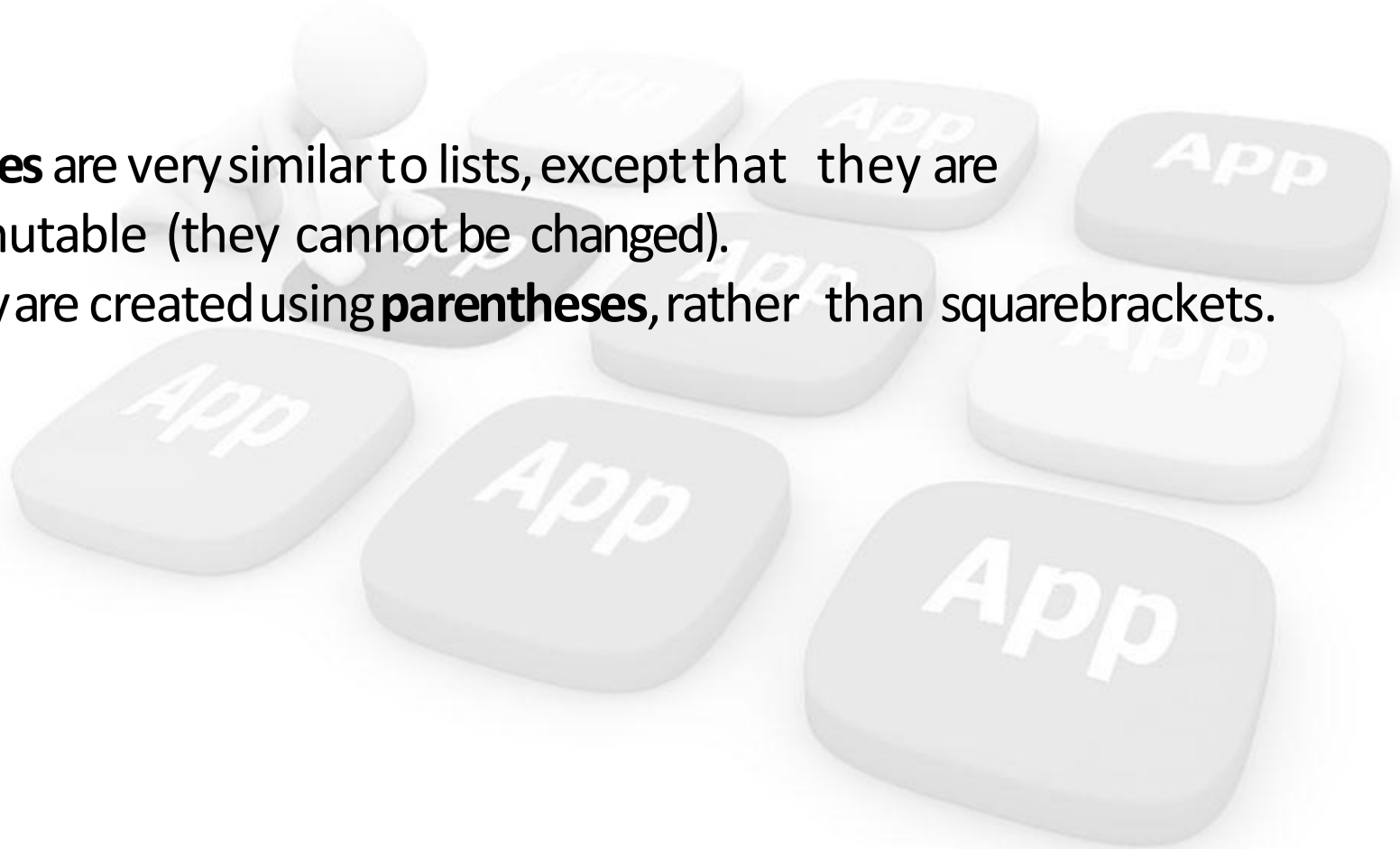
# Python Tuples

- **Tuples** are very similar to lists, except that they are immutable (they cannot be changed).
- They are created using **parentheses**, rather than square brackets.

- We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

# Creating a Tuple

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma.
- The parentheses are optional but is a good practice to write it.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

```python
# empty tuple
my_tuple = ()
print(my_tuple)

# tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)

# tuple can be created without parentheses
# also called tuple packing
my_tuple = 3, 4.6, "dog"
print(my_tuple)
```

```
()
(1, 2, 3)
(1, 'Hello', 3.4)
('mouse', [8, 4, 6], (1, 2, 3))
(3, 4.6, 'dog')
```

- Creating a tuple with one element is a bit tricky.
- Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is in fact a tuple.

```python
# only parentheses is not enough
my_tuple = ("hello")
print(type(my_tuple))

# need a comma at the end
my_tuple = ("hello",)
print(type(my_tuple))

# parentheses is optional
my_tuple = "hello",
print(type(my_tuple))
```

```
<class 'str'>
<class 'tuple'>
<class 'tuple'>
```

# Accessing Elements in a Tuple

You can access the values in the tuple with their index, just as you did with  lists:

* Nested tuple are accessed using nested indexing
* Negative indexing can be applied to tuples similar to lists.
* We can access a range of items in a tuple by using the slicing operator
* Trying to reassign a value in a tuple causes a  TypeError.

```
marks = (23,45,32)
print(marks[0])
print(marks[2])
```

```
23
32
```

```
# nested tuple
n_tuple = ("SIKANDER", [8, 4, 6], (1, 2, 3))

print(n_tuple[0])
print(n_tuple[1])

print(n_tuple[0][0])
print(n_tuple[1][0])
```

```
SIKANDER
[8, 4, 6]
S
8
```

# Changing a Tuple

- Unlike lists, tuples are immutable.
- This means that elements of a tuple cannot be changed once it has been  assigned. But, if the element is itself a mutable datatype like list, its nested  items can be changed.

```
n_tuple = ("SIKANDER", [8, 4, 6], (1, 2, 3))
print(n_tuple)

n_tuple[1][1] = 23
print(n_tuple)
```

```
('SIKANDER', [8, 4, 6], (1, 2, 3))
('SIKANDER', [8, 23, 6], (1, 2, 3))
```

Similar to List,

- We can use + operator to combine two tuples. This is also called **concatenation**.
- We can also **repeat** the elements in a tuple for a given number of times using the * operator.
- Both + and * operations result into a new tuple.

```
# Concatenation
print((1, 2, 3) + (4, 5, 6))

# Repeat
print(("Repeat",) * 3)

(1, 2, 3, 4, 5, 6)
('Repeat', 'Repeat', 'Repeat')
```

# Deleting a Tuple

- We cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple.
- But deleting a tuple entirely is possible using the keyword del.

```
my_tuple = ('p','r','o','g','r','a','m','i','z')

del my_tuple[3]
# TypeError: 'tuple' object doesn't support item deletion

# can delete entire tuple
del my_tuple

# NameError: name 'my_tuple' is not defined
my_tuple
```

# Python Tuple Methods

- Methods that add items or remove items are not available with tuple.
  Only the following two methods are available.

| Method | Description |
|---|---|
| count(x) | Return the number of items that is equal to x |
| index(x) | Return index of first item that is equal to x |

```python
my_tuple = ('a','p','p','l','e',)

print('Total count of element p is ' , my_tuple.count('p'))

print('Index of l is' , my_tuple.index('l'))
```

```
Total count of element p is  2
Index of l is 3
```
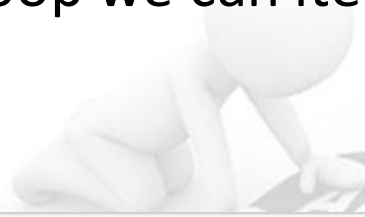
# Tuple Membership Test

- We can test if an item exists in a tuple or not, using the keyword in.

```python
my_tuple = ('a','p','p','l','e',)

print('a' in my_tuple)

print('b' in my_tuple)

print('g' not in my_tuple)
```

```
True
False
True
```

# Iterating Through a Tuple

Using a for loop we can iterate though each item in a tuple.

```python
names = ('Sikander', 'Sharath','John','Kate')
for name in names:
    print('Hello ',name)
```

```
Hello  Sikander
Hello  Sharath
Hello  John
Hello  Kate
```

# Built-in Functions with Tuple

| Function | Description |
|----------|-------------|
| 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. |