At the end of this lecture, students should be able to:

- understand the tuple type
- create tuples
- manipulate code which contains tuples
- return tuples
- know the differences between tuples and lists
Recap

- From lecture 16
  - Understand various operators on lists
  - Understand various list methods
  - Lists are mutable

```python
def main():
    numbers = [1206, 1216, 475, 1038, 1481, 135]
    search_numbers = [1038, 1367, 1206, 740, 1281, 1216]
    index_list = get_index_list(numbers, search_numbers)
    print("indices of the numbers found", index_list)

def get_index_list(number_list, numbers_to_look_for):
    a_list = list()
    for number in numbers_to_look_for:
        if number in number_list:
            index = number_list.index(number)
            a_list.append(index)
    a_list.sort()
    return a_list

main()
```

indices of the numbers found [0, 1, 3]
Creating Tuples

- **Tuples**
  - The items of a tuple are enclosed inside a pair of parentheses, separated by commas even if there is only one item in the tuple
  - A tuple is an ordered sequence of items of any types
  - Tuples are sequences - the elements of a tuple have an order.

- **Create Tuples**
  - an empty tuple (empty tuple is not useful – find out why later in these slides)

- Other tuple creation examples:

  ```python
  tuple1 = ()  # or tuple1 = tuple()
  tuple2 = ("red", "blue")
  tuple3 = (1, 2, "jim")
  tuple4 = (3,)
  Needs the comma if only one element in the tuple
  ```
Creating Tuples with a single element

- **Create Tuples**
  - If the tuple contains a single element, a comma is added after the element. This is required as, otherwise, there would be confusion between a tuple and a parenthesised object, e.g., (5) is the same as the integer 5).

```python
tuple1 = (5,)
```

```python
print( type((5,)) )  # (5,) is a tuple
print( type((5)) )   # (5) is an int (with parentheses around it)
```

```
<class 'tuple'>
<class 'int'>
```

Note that `tuple` is a built-in type and it should not be used as a variable name.
Printing the elements of tuples

Use the print() function to print the elements of the tuple.

tuple1 = (3, 6, 8)
print(tuple1)

tuple2 = ("abcdef", "ghij", "klmno")
print(tuple2)

(3, 6, 8)
('abcdef', 'ghij', 'klmno')

Note that tuples use round brackets.
Accessing the elements of a tuple

- Accessing tuple elements
  - Each element in a tuple can be accessed using the index value (starting from index 0) and square brackets.
  - The elements of a tuple can be accessed from the end of the tuple backwards using a negative index.
  - for ... in loop can be used to visit each element of a tuple (iterate through the tuple).

```python
tuple1 = (3, 6, 8)
print("1.", tuple1[1], tuple1[2], tuple1[0])
print("2.", tuple1[-2], tuple1[-3])

tuple2 = (tuple1[1], tuple1[2], tuple1[0], 1, 7)
print("3.", tuple2)
print()

for element in tuple2:
    if element > 3:
        print(element)
```

1. 6 8 3
2. 6 3
3. (6, 8, 3, 1, 7)

6
8
7
+, * and in operators and tuples

- Operators + (concatenate), * (repeat), and in (membership) can be used with tuples. The result of the + and * operators is a new tuple object.

```python
tuple1 = (3, 6, 8)
tuple2 = (5, 1, 0, 4)
tuple3 = tuple2 + tuple1
tuple4 = tuple1 * 3

print(0 in tuple1)
print(0 in tuple2)
print("1.", tuple1)
print("2.", tuple2)
print("3.", tuple3)
print("4.", tuple4)
```

```
False
True
1. (3, 6, 8)
2. (5, 1, 0, 4)
3. (5, 1, 0, 4, 3, 6, 8)
4. (3, 6, 8, 3, 6, 8, 3, 6, 8)
```
Tuples can be sliced in the same way as strings and lists are sliced. The result is a new tuple.

tuple1 = (3, 6, 8, 0, 1, 2, 7)

print("1.", tuple1[0:6:2])
print("2.", tuple1[2:7:3])
print("3.", tuple1[5:1:-1])

1. (3, 8, 1)
2. (8, 2)
3. (2, 1, 0, 8)
Tuples are immutable

- Tuples are "immutable", i.e., the elements of a tuple object cannot be changed.

```python
tuple1 = (3, 6, 8)
tuple2 = tuple1
tuple3 = (tuple2[0], tuple2[1], tuple2[2])

print("1.", tuple1 is tuple2)
tuple1 = tuple1 + (5,)

print("2.", tuple1)
print("3.", tuple2)

print("4.", tuple1 is tuple2)
print("5.", tuple2 == tuple3)
print("6.", tuple2 is tuple3)
```

1. True
2. (3, 6, 8, 5)
3. (3, 6, 8)
4. False
5. True
6. False
def fiddle_tuples(t1, t2):
    t3 = (t1[1], t2[0])
    t1 = (t3[1], t2[0])
    t2 = t1
    return t3

def main():
    t1 = (3, 5)
    t2 = (4, 7)
    t3 = t2
    t3 = fiddle_tuples(t1, t2)
    print("t1:", t1, "t2:", t2, "t3:", t3)

main()
Converting tuples into lists

- The shortcut way of creating an empty list is
- The alternative way of creating an empty list is

- A tuple can be converted into a list by enclosing the tuple inside `list(...)`, i.e., passing the tuple as an argument.

```python
tuple1 = (3, 6, 8)
a_list = list(tuple1)
```

```
tuple1 = (3, 6, 8, 9, 5)
a_list = list(tuple1)
a_list.sort()
```

```
print("1.", tuple1)
print("2.", a_list)
```

1. (3, 6, 8, 9, 5)
2. [3, 5, 6, 8, 9]
Converting lists into tuples

- The shortcut way of creating an empty tuple is
  ```python
  a_tuple = ()
  ```

- The alternative way to create an empty tuple is
  ```python
  a_tuple = tuple()
  ```

- A list can be converted into a tuple by enclosing the list inside `tuple(...)` , i.e., passing the list as an argument.
  ```python
  a_list = [3, 6, 8]
  a_tuple = tuple(a_list)
  ```

```python
tuple1 = (3, 6, 8, 2, 5)
a_list = list(tuple1)
a_list.sort()
a_tuple = tuple(a_list)

print("1.", a_list)
print("2.", tuple1)
print("3.", a_tuple)
```

1. [2, 3, 5, 6, 8]
2. (3, 6, 8, 2, 5)
3. (2, 3, 5, 6, 8)
Multiple assignment

Assignment to more than one variable can be done on ONE line.

```python
scores = (56, 78, 91)
(test1, test2, test3) = scores  # or test1, test2, test3 = scores
name1, name2, name3 = "Bob", "Jane", "Jill"
name2 = name2 + "-marie"

print("1.", test2, test1, test3)
print("2.", name3, name1, name2)
```

1. 78 56 91
2. Jill Bob Jane-marie
Returning more than one value

- Functions can return a tuple of values.

```python
def get_a_date():
    months = ("January", "February", ..., "November", "December")
    days_in_month = (31, 28, 31, 30, 31, ..., 30, 31, 30, 31)
    days = ("Sunday", "Monday", ..., "Saturday")
    day_number = random.randrange(0, len(days))
    month_number = random.randrange(0, len(months))
    date = random.randrange(1, days_in_month[month_number] + 1)
    return (days[day_number], months[month_number], date)

def main():
    date = get_a_date()
    print("Your best day next year is a", date[0], "around",
          date[1], date[2])
    date = get_a_date()
    print("Next year be careful on a", date[0], "around",
          date[1], date[2])
main()
```

Your best day next year is a Wednesday around February 14
Next year be careful on a Sunday around November 10
A tuple function

- **index(x)** returns the index of the first element from the left in the tuple with a value equal to x.

- Python throws an error if there is no such value in the list. Because of this, index(x) is usually preceded by a check for that element using the **in** operator.

```python
tuple1 = (10, 20, 30, 40, 50, 55)
if 40 in tuple1:  # check first
    position = tuple1.index(40)
    print("40 is in position", position, "in the tuple")
else:
    print("40 is not in the tuple")
```

40 is in position 3 in the tuple
Exercise

Complete the `get_uniques_tuple()` function which returns a tuple made up of all the unique values in the parameter tuple, `a_tuple`. You may find that you need to work with a list, and, finally, convert the list to a tuple.

```python
def get_uniques_tuple(a_tuple):

def main():
    a_tuple = get_uniques_tuple((3, 4, 5, 6, 3, 2, 9, 4, 5, 6, 2, 9))
    print("Without duplicates", a_tuple)

main()
```

Without duplicates (3, 4, 5, 6, 2, 9)
Exercise

- Complete the `carry_out_transactions()` function which is passed an initial balance and a tuple of transactions (positive and negative amounts). The function returns a tuple made up of three values: the final balance, the sum of all the deposits and the sum of all the withdrawals.

```python
def carry_out_transactions(balance, transactions_tuple):
    # Implementation goes here

def main():
    results = carry_out_transactions(5400, (100, -400, 500, -800, 600, -100, -200, 50, 0, -200))
    print("Balance $", results[0], ", deposits $", results[1], ", withdrawals $", results[2], sep="")
main()
```

Balance $4950, deposits $1250, withdrawals $1700
Why tuples?

- Tuples cannot be inadvertently changed (remember they are immutable). They are a useful tool if you want to use read-only information.
- Tuples are immutable and can be used where only immutable objects can be used (this becomes important later in course).
- Processing tuples is faster than processing lists.
- Assignment to multiple variables (packed inside a tuple) can be done on the same line of code.
- A function can return multiple values (packed inside a tuple).

- Tuples are processed more quickly than lists. If you are not going to change the elements of a series of objects, use a tuple rather than a list.
A tuple stores data as a sequence

- The operators: +, *, and in can be used with tuples
- We use a for ... in ... to iterate through the contents of a tuple
- len() returns the number of elements in a tuple
- min() returns the minimum of the elements in a tuple
- max() returns the maximum of the elements in a tuple
- sum() returns the sum of the elements in a tuple
- Each element of the tuple can be accessed using the index operator. The index can be negative (starting from the end of the tuple)
- Slices of tuple can be obtained by using [slice_start: slice_end: step]
- Tuples are immutable and therefore the elements of a tuple can be accessed but not changed

- Tuples can be converted into lists and vice versa
- Assignment to multiple variables (packed inside a tuple) can be done on the same line of code.
- A function can return multiple values (packed inside a tuple).
tuple1 = (5, 7, 2, 6, 4, 3, 9)
tuple2 = (6, )
for element in tuple1:
    ...

number_of_elements = len(tuple1)
min_value = min(tuple1)
max_value = max(tuple1)
total = sum(tuple1)
element_from_end = tuple1[-2]
tuple2 = tuple1[1:5:2]
position = tuple1.index(3)
tuple3 = tuple([8, 4, 9])
list1 = list(tuple1)
(a, b, c) = ("ant", "bee", "cat")
def get_results():
    return (56, 23, 91)