

# COMPSCI 1😊1

Principles of Programming

Lecture 18 - Tuples

# Learning outcomes

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

- Understand various operators on lists
- Understand various list methods
- Lists are mutable

```
def remove_multiples(number_list, multiples_of):
    for index in range(len(number_list)-1, -1, -1):
        if number_list[index] % multiples_of == 0:
            number_list.pop(index)

def main():
    numbers = [25, 5, 9, 10, 15, 8]
    print(numbers)
    remove_multiples(numbers, 5) #remove multiples of 5
    print("Numbers left", numbers)

main()
```

```
[25, 5, 9, 10, 15, 8]
Numbers left [9, 8]
```

# 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

```
tuple1 = ()
```

or

```
tuple1 = tuple()
```

- Other tuple creation examples:

```
tuple1 = (4, True, "Test", 34.8)
```

```
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).

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

```
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 tuples

## 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).

```
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.

```
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)



# Slicing tuples

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.

```
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

# Converting tuples into lists

The shortcut way of creating an empty list is:

```
a_list = []
```

The alternative way of creating an empty list is:

```
a_list = list()
```

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

```
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:

```
a_tuple = ()
```

The alternative way to create an empty tuple is:

```
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.

```
a_list = [3, 6, 8]  
a_tuple = tuple(a_list)
```

```
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.

```
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.

```
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 method

**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.

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

```
40 is in position 3 in the tuple
```







# 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.

# Summary

## 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).

# Python features used in this lecture

```
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)
```