Session 2

Lists

Lists are used to store multiple items in a single variable.
Lists are created using square brackets:

```python
fruit = ["apple", "banana", "cherry"]
print(fruit)
```

List Items

List items are **ordered**, **changeable** and **allow duplicate** values.
List items are **indexed**, **the *first* item has index** `[0]`, the *second* item has index `[1]` etc.

Access Items

List items are indexed and you can access them by referring to the index number:

```python
fruit = ["apple", "banana", "cherry"]
print(fruit[1])
```

Negative Indexing

Negative indexing means start from the end. `-1` refers to the last item, `-2` refers to the second last item etc.

```python
fruit = ["apple", "banana", "cherry"]
print(fruit[-1])
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.
When specifying a range, the return value will be a new list with the specified items.

```python
fruit = ["apple", "banana", "cherry", "orange", "kiwi", "melon",
"mango"]
print(fruit[2:5])
```

> **Note:** The search will start at index 2 (included) and end at index 5 (not included).

By leaving out the start value, the range will start at the first item:

```python
fruit = ["apple", "banana", "cherry", "orange", "kiwi", "melon",
"mango"]
print(fruit[:4])
```

By leaving out the end value, the range will go on to the end of the list:

```python
fruit = ["apple", "banana", "cherry", "orange", "kiwi", "melon",
"mango"]
print(fruit[2:])
```

Check if Item Exists

To determine if a specified item is present in a list use the `in` keyword:

```python
fruit = ["apple", "banana", "cherry"]
if "apple" in fruit:
  print("Yes, 'apple' is in the fruits list")
```

Change Item Value

To change the value of a specific item, refer to the index number:

PYTHON

```python
fruit = ["apple", "banana", "cherry"]
fruit[1] = "blackcurrant"
print(fruit)
```

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

PYTHON

```python
# Change the values "banana" and "cherry" with the values
"blackcurrant" and "watermelon":
fruit = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
fruit[1:3] = ["blackcurrant", "watermelon"]
print(fruit)
```

Append Items

To add an item to the end of the list, use the append() method:

PYTHON

```python
fruit = ["apple", "banana", "cherry"]
fruit.append("orange")
print(fruit)
```

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the `insert()` method.

The `insert()` method inserts an item at the specified index:

PYTHON

```python
fruit = ["apple", "banana", "cherry"]
fruit.insert(2, "watermelon")
print(fruit)
```

Extend List

To append elements from *another list* to the current list, use the `extend()` method.

```python
fruit = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
fruit.extend(tropical)
print(fruit)
```

Remove Specified Item

The `remove()` method removes the specified item.

```python
fruit = ["apple", "banana", "cherry"]
fruit.remove("banana")
print(fruit)
```

Remove Specified Index

The `pop()` method removes the specified index.

```python
fruit = ["apple", "banana", "cherry"]
fruit.pop(1)
print(fruit)
```

> If you do not specify the index, the `pop()` method removes the **last item**.

List Methods

Python has a set of built-in methods that you can use on lists

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position

Method	Description
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Tuple

Tuples are used to store multiple items in a single variable.
A tuple is a collection which is ordered and **unchangeable**. Tuples are written with round brackets.

PYTHON

```python
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

PYTHON

```python
thistuple = ("apple",)
print(type(thistuple))

# NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets (same as lists):

PYTHON

```python
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Dictionary

Dictionaries are used to store data values in *key : value* pairs.
A dictionary is a collection which is ***ordered**, **changeable** and **do not allow duplicates**.

> As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

PYTHON

```python
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(car)
```

Dictionary Items - Data Types

The values in dictionary items can be of any data type:

PYTHON

```python
# String, int, boolean, and list data types:
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

Accessing Items

You can access the items of a dictionary by referring to its **key** name, inside square brackets:

PYTHON

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

Get Keys

The `keys()` method will return a list of all the keys in the dictionary.

PYTHON

```python
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x)
```

Get Values

The `values()` method will return a list of all the values in the dictionary.

PYTHON

```python
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x)
```

Change Values

You can change the value of a specific item by referring to its key name:

PYTHON

```python
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car["year"] = 2018
```

Update Dictionary

The `update()` method will update the dictionary with the items from the given argument.

PYTHON

```python
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car.update({"year": 2020})
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

PYTHON

```python
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car["color"] = "red"
print(car)
```

you can also use `update()` method

Removing Items

The `pop()` method removes the item with the specified key name:

PYTHON

```python
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car.pop("model")
print(car)
```

The `clear()` method empties the dictionary:

PYTHON

```python
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car.clear()
print(car)
```

Python Loops

Python has two primitive loop commands:

- `while` loops
- `for` loops

The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

PYTHON

```python
# Print i as long as i is less than 6:
i = 1
while i < 6:
  print(i)
  i += 1
```

> **Note:** remember to increment i, or else the loop will continue forever.

The break Statement

With the break statement we can stop the loop even if the while condition is true:

PYTHON

```python
# Exit the loop when i is 3:
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

The For Loops

A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the `for` keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the `for` loop we can execute a set of statements, once for each item in a list, tuple, set etc.

PYTHON

```python
# Print each fruit in a fruit list:
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

The break Statement

With the `break` statement we can stop the loop before it has looped through all the items:

PYTHON

```python
# Exit the loop when `x` is "banana":
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

The range() Function

To loop through a set of code a specified number of times, we can use the `range()` function,

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

PYTHON

```python
for x in range(6):
  print(x)
```

An Example

here is a simple username and password checker:

PYTHON

```python
user = {
    'username': 'matin007',
    'password': '123456'
}
status = True

while status:
    entered_username = input('please inter your username: ')
    entered_pasword = input('please inter your password: ')
    if (entered_pasword == user['password']) and (entered_username
== user['username']) :
        status = False
        print('you are loged in.')
    else:
        print('the password or username is incorrect.')
```