LISTS

1. Creating a list
2. Accessing a list item
3. Changing a list item
4. Getting the length of a list
5. Creating a sub-list from existing list
6. Concatenating lists
7. Checking if an item is in a list
8. List methods
9. Iterating a list

1. Creating a list

```
# Creating a list and storing it in the variable names
names = [‘Nicole’, ‘Kayleigh’, ‘Adora’, ‘Valerie’, ‘Paige’, ‘Thejesvi’]

# A list can contain different data types
details = [‘Nicole’, 15, ‘+65 8786 2525’, True, 37.2]

# Shortcut for creating a list containing the same content:
# This creates a list containing 5 items, and all of them are zeros i.e. [0, 0, 0, 0, 0]
scores = [0] * 5

# This creates a list containing 10 items, and all of them are the strings ‘Not Available’
names = [‘Not Available’]*10

# Creates an empty list
spam = []

# Creates an empty list
eggs = list()
```

2. Accessing a list item

```
# The list is indexed (numbered). The index starts with 0 for the first item.

# Accessing the first item
names = [‘Nicole’, ‘Kayleigh’, ‘Adora’, ‘Valerie’, ‘Paige’, ‘Thejesvi’]
names[0]       # returns ‘Nicole’, but you do not see this since it is not printed
print(name[0]) # prints Nicole to the terminal

names[3]       # ‘Valerie’
names[5]       # ‘Thejesvi’
names[-1]      # ‘Thejesvi’
names[-4]      # ‘Adora’

names[6]       # IndexError: index out of range
```

3. Changing a list item

```
names = [‘Nicole’, ‘Kayleigh’, ‘Adora’, ‘Thejesvi’]

names[-1] = ‘Ramesh’   # names is now [‘Nicole’, ‘Kayleigh’, ‘Adora’, ‘Ramesh’]

names[2] = ‘Deborah’   # names is now [‘Nicole’, ‘Kayleigh’, ‘Deborah’, ‘Ramesh’]
```

4. Getting the length of a list

```
# The len() function returns the length of
# a sequence (i.e. string, bytes, tuple, list or range) or
# a collection (i.e. dictionary, set, frozen set)

names = [‘Nicole’, ‘Kayleigh’, ‘Adora’, ‘Valerie’, ‘Paige’, ‘Thejesvi’]
len(names)     # 6

details = [‘Nicole’, 15, ‘+65 8786 2525’, True, 37.2]
len(details)   # 5

scores = [0] * 50
len(scores)    # 50

empty = []
len(empty)     # 0
```

5. Creating a sub-list from existing list

```
names = [‘Nicole’, ‘Kayleigh’, ‘Adora’, ‘Valerie’, ‘Paige’, ‘Thejesvi’]
names[2:5]     # returns sub-list starting from index 2 of the list to 5 exclusive i.e.
               # [‘Adora’, ‘valerie’, ‘Paige’]

# Sub-list from the beginning of the list to index 3 exclusive
names[:3]      # [‘Nicole’, ‘Kayleigh’, ‘Adora’]

# Sub-list from index 4 to the end of the list
names[4:]      # [‘Paige’, ‘Thejesvi]

# Sub-list from the beginning of the list,
# to a random index of 0 – 6 (exclusive) i.e. 0 - 5
from random import randrange
names[:randrange(len(names))]
```

6. Concatenating lists

```
cat_persons = [‘Nancy’, ‘Josiah’, ‘Peter’]
dog_persons = [‘Angie’, ‘Julia’, ‘Desmond’]

# Concatenates list cat_persons with dog_persons and assign the resulting list
# to the variable cat_dog_group. cat_dog_group is now the list:
# [‘Nancy’, ‘Josiah’, ‘Peter’, ‘Angie’, ‘Julia’, ‘Desmond’]
cat_dog_group = cat_persons + dog_persons

[88, 99] + [77]        # [88, 99, 77]
```

7. Checking if an item is in a list

```
names = [‘Nicole’, ‘Kayleigh’, ‘Adora’, ‘Valerie’, ‘Paige’, ‘Thejesvi’]

‘Kayleigh’ in names    # True
‘kayleigh’ in names    # False

# Prints ‘Thomas is not here.’
person = ‘Thomas’
if person in names:
    print(person, ‘is here.’)
else:
    print(person, ‘is not here.’)
```

8. List methods

Method	Description	Returns
append(object)	Appends object to the end of the list	None
clear()	Removes all items from the list	None
copy()	Returns a shallow copy of the list	List
extend(iterable)	Extends list by appending elements from the iterable.	None
index(value, [start, [stop]])	Returns first index of value. Raises ValueError if value is not present. Option to specify start and stop indexes.	Integer
insert(index, object)	Insert object before the index.	None

8. List methods

Method	Description	Returns
pop([index])	Removes and returns items at the index (defaulted to last index). Raises IndexError if list is empty or index is out of range.	item
remove(value)	Removes first occurrence of value. Raises ValueError if the value is not present.	None
reverse()	Reverses the list in place.	None
sort(key=None, reverse=False)	Sorts the list in place. Stable sort.	None

* A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted. Some sorting algorithms are stable by nature like Insertion sort, Merge Sort, Bubble Sort, etc.

8. List methods

```
names = ['Nancy', 'Josiah', 'Peter']

names.append('Joy')    # names is now ['Nancy', 'Josiah', 'Peter', 'Joy']
                       # Experiment: What happens if you append a list e.g. ['Joy', 'Jojo']
                       # instead of the string 'Joy'?

popped = names.pop()   # names is now ['Nancy', 'Josiah', 'Peter']
                       # popped contains 'Joy'

names.extend(['Joy', 'Jojo']) # names is now ['Nancy', 'Josiah', 'Peter', 'Joy', 'Jojo']
                              # Think: What's the difference between the extend method
                              # and the + list concatenation operator?

duplicate = names.copy()      # duplicate is now ['Nancy', 'Josiah', 'Peter', 'Joy', 'Jojo']
                              # copy() method does a shallow copy. Find out what shallow
                              # copy means (vs deep copy).

duplicate.clear()      # duplicate is now []
```

8. List methods

```
# Continued from previous page
# names is currently [‘Nancy’, ‘Josiah’, ‘Peter’, ‘Joy’, ‘Jojo’]

names.index(‘Joy’)     # 3
names.index(‘Joyous’)  # ValueError: ’Joyous’ is not in the list
names.index(‘joy’)     # ValueError: ’joy’ is not in the list

names.index(‘Joy’, 2)  # 3
                       # This means search for ‘Joy’ in the list from index 2 onwards
                       # and return the index for the first occurrence of ‘Joy’

names.index(‘Joy’, 0, 3)      # ValueError: ’Joy’ is not in the list
                              # This means search for ‘Joy’ in the list from index 0
                              # to index 3 (exclusive).

names.index(‘Joy’, 2, 4)      # 3
```

8. List methods

```
# Continued from previous page
# names is currently [‘Nancy’, ‘Josiah’, ‘Peter’, ‘Joy’, ‘Jojo’]

names.remove(‘Josiah’)# names is now [‘Nancy’, ‘Peter’, ‘Joy’, ‘Jojo’]

names.insert(2, ‘Thomas’)     # names is now [‘Nancy’, ‘Peter’, ‘Thomas’, ‘Joy’, ‘Jojo’]
                              # Experiment: What happens if you insert beyond
                              # the bounds of the list index?

names.insert(1, names.pop(2)) # names is now [‘Nancy’, ‘Thomas’, ‘Peter’, ‘Joy’, ‘Jojo’]

names.sort()   # names is now [‘Jojo’, ‘Joy’, ‘Nancy’, ‘Peter’, ‘Thomas’]

names[-1] = ‘Tom’
names.sort(key=len)    # Sort names using the len() function as key argument. This means
                       # that the list will be sorted based on the value returned when
                       # len() function is applied to each item in the list i.e. names be
                       # sorted by length of each string in it.
                       # names is now [‘Joy’, ‘Tom’, ‘Jojo’, ‘Nancy’, ‘Peter’]
```

8. List methods

```
# Continued from previous page
# names is currently ['Joy', 'Tom', 'Jojo', 'Nancy', 'Peter']


names.reverse()         # names is now ['Peter', 'Nancy', 'Jojo', 'Tom', 'Joy']

names.sort()            # names is now ['Jojo', 'Joy', 'Nancy', 'Peter', 'Tom']

names.reverse()         # names is now ['Tom', 'Peter', 'Nancy', 'Joy', 'Jojo']
```

9. Iterating a list

```
# The for loop can be used to iterate through a list

from random import randrange

adjectives = [‘delicious’, ‘healthy’, ‘appetizing’]
veg = [‘Broccoli’, ‘Spinach’, ‘Bok choy’, ‘Carrot’]

def adjective():
    return ‘ is ’ + adjectives[randrange(len(adjectives))]

for thing in veg:
    print(thing + adjective())

# Experiment: test this on your Python shell or by running a Python file
# Possible result:
    Broccoli is appetizing
    Spinach is healthy
    Bok choy is appetizing
    Carrot is delicious
```