Artificial Intelligence ¶

LAB Two

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

In [1]:
```
a = "Hello"
print(a)
```

```
Hello
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example You can use three double quotes:

In [2]:
```
a = """Python is a popular object-oriented programing
language having the capabilities
of highlevel programming language."""
print(a)
```

```
Python is a popular object-oriented programing
language having the capabilities
of highlevel programming language.
```

Or three single quotes:

Example

In [3]:
```
a = '''Python is a popular object-oriented programing
language having the capabilities
of highlevel programming language.'''
print(a)
```

```
Python is a popular object-oriented programing
language having the capabilities
of highlevel programming language.
```

Note: in the result, the line breaks are inserted at the same position as in the code.

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

In [4]:
```
a = "Hello, World!"
print(a[1])
```

```
e
```

String Length

To get the length of a string, use the len() function.

Example

The len() function returns the length of a string:

In [5]:
```
a = "Hello, World!"
print(len(a))
```

```
13
```

Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

Example

Check if "free" is present in the following text:

In [6]:
```
txt = "The best things in life are free!"
print("free" in txt)
```

```
True
```

Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

Example

Check if "expensive" is NOT present in the following text:

In [8]:
```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

```
True
```

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

In [9]:
```
b = "Hello, World!"
print(b[2:5])
```

```
llo
```

Note: The first character has index 0.

Slice From the Start

By leaving out the start index, the range will start at the first character:

Example

Get the characters from the start to position 5 (not included):

In [10]:
```
b = "Hello, World!"
print(b[:5])
```

```
Hello
```

Slice To the End

By leaving out the end index, the range will go to the end:

Example

Get the characters from position 2, and all the way to the end:

In [11]:
```
b = "Hello, World!"
print(b[2:])
```

```
llo, World!
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

In [12]:
```
b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

Python - Modify Strings

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

Upper Case

Example

The upper() method returns the string in upper case:

In [13]:
```
a = "Hello, World!"
print(a.upper())
```

```
HELLO, WORLD!
```

Lower Case

Example

The lower() method returns the string in lower case:

In [14]:
```
a = "Hello, World!"
print(a.lower())
```

```
hello, world!
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

The strip() method removes any whitespace from the beginning or the end:

In [15]:
```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

```
Hello, World!
```

Replace String

Example

The replace() method replaces a string with another string:

In [16]:
```
a = "Hello, World!"
print(a.replace("H", "J"))
```

```
Jello, World!
```

Split String

The split() method returns a list where the text between the specified separator becomes the list items.

Example

The split() method splits the string into substrings if it finds instances of the separator:

In [17]:
```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

```
['Hello', ' World!']
```

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable a with variable b into variable c:

In [18]:
```
a = "Hello"
b = "World"
c = a + b
print(c)
```

```
HelloWorld
```

Example

To add a space between them, add a " ":

In [19]:
```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

```
Hello World
```

String Format

As we learned in the Python that we cannot combine strings and numbers like this:

Example

In [ ]:
```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Example

Use the format() method to insert numbers into strings:

In [21]:
```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

```
My name is John, and I am 36
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

In [22]:
```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example

In [23]:
```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567.
```

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

In [ ]:
```
Example
You will get an error if you use double quotes inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."
```

To fix this problem, use the escape character \ ":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

In [24]:
```
txt = "We are the so-called \"Vikings\" from the north."

print(txt)
```

```
We are the so-called "Vikings" from the north.
```

built-in data types

There are four collection data types in the Python programming language:

- **List:** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple:** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set:** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary:** is a collection which is ordered** and changeable. No duplicate members.

1. List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

Example Create a List:

In [26]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

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.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

Note: There are some list methods that will change the order, but in general: the order of the items will not change.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

Example

Lists allow duplicate values:

In [27]:
```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry', 'apple', 'cherry']
```

List Length

To determine how many items a list has, use the len() function:

Example

Print the number of items in the list:

In [28]:
```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

```
3
```

List Items - Data Types

List items can be of any data type:

Example

String, int and boolean data types:

In [29]:
```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

A list can contain different data types:

Example

A list with strings, integers and boolean values:

In [30]:
```
list1 = ["abc", 34, True, 40, "male"]
```

type()

From Python's perspective, lists are defined as objects with the data type 'list':

Example

What is the data type of a list?

In [31]:
```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

```
<class 'list'>
```

The list() Constructor

It is also possible to use the list() constructor when creating a new list.

Example

Using the list() constructor to make a List:

In [32]:
```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

Access Items

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

Example

Print the second item of the list:

In [33]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
banana
```

Note: The first item has index 0.

Negative Indexing

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

In [34]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

```
cherry
```

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.

Example

Return the third, fourth, and fifth item:

In [35]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

```
['cherry', 'orange', 'kiwi']
```

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

Remember that the first item has index 0.

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

Example

This example returns the items from the beginning to, but NOT including, "kiwi":

In [36]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

```
['apple', 'banana', 'cherry', 'orange']
```

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

Example

This example returns the items from "cherry" to the end:

In [37]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

```
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

In [38]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

```
['orange', 'kiwi', 'melon']
```

Change List Items

Change Item Value

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

Example

Change the second item:

In [39]:
```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

Change a Range of Item Values

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:

Example

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

In [40]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

If you insert more items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second value by replacing it with two new values:

In [41]:
```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'blackcurrant', 'watermelon', 'cherry']
```

Note: The length of the list will change when the number of items inserted does not match the number of items replaced.

If you insert less items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second and third value by replacing it with one value:

In [ ]:
```

```