Part 4

Introduction to Programming

26.9.2024

Last week

- Repetition with while
- break and continue
- Nested loops
- Indexing strings
- Extracting characters and substrings
- Introduction to own functions

Parameters and arguments

Parameter is the variable defined in the function definition

Argument is the value passed to the function when it's called

```
def greet(name):
    print("Hello there,", name)

def sum(a, b):
    print("The sum of the arguments is", a + b)
```

```
greet("Emily")
sum(2, 3)
```

Return value

Function can return a value with **return** statement

```
number = int(input("Please type in an integer: "))
```

This way function call can be used as part of an expression

return and print

Notice the difference between **return** and **print**

```
def max1(a, b):
    if a > b:
        return a
    else:
        return b

def max2(a, b):
    if a > b:
        print(a)
    else:
        print(b)

result = max1(3, 5)
print(result)

max2(7, 2)
```

Type hints

Type	Python data type	Example
integer	`int`	`23`
floating point number	`float`	`-0.45`
string	`str`	`"Peter Python"`
Boolean value	`bool`	`True`

List

List is a collection of homogeneous items

```
my_list = []
```

New list can be created with bracket notation

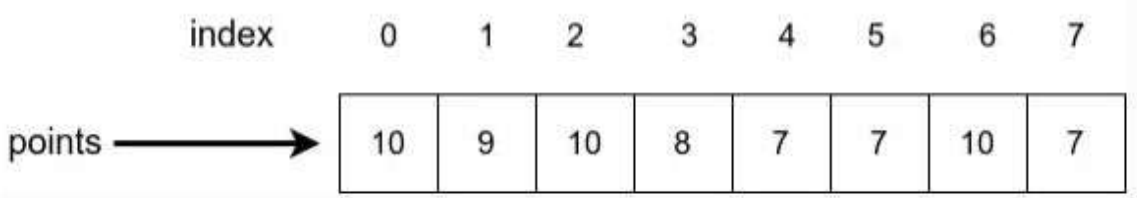
```
my_list = [7, 2, 2, 5, 2]
```

Indexing items

Items are indexed similarly to characters in a string

First item at index zero

```
points = [10, 9, 10, 8, 7, 7, 10, 7]
```



Length of a list

The length of a list can be returned with **len** function

```
my_list = [7, 2, 2, 5, 2]
print(len(my_list))
```

Again, this works similarly to strings: it returns the number of items in a list

Adding items

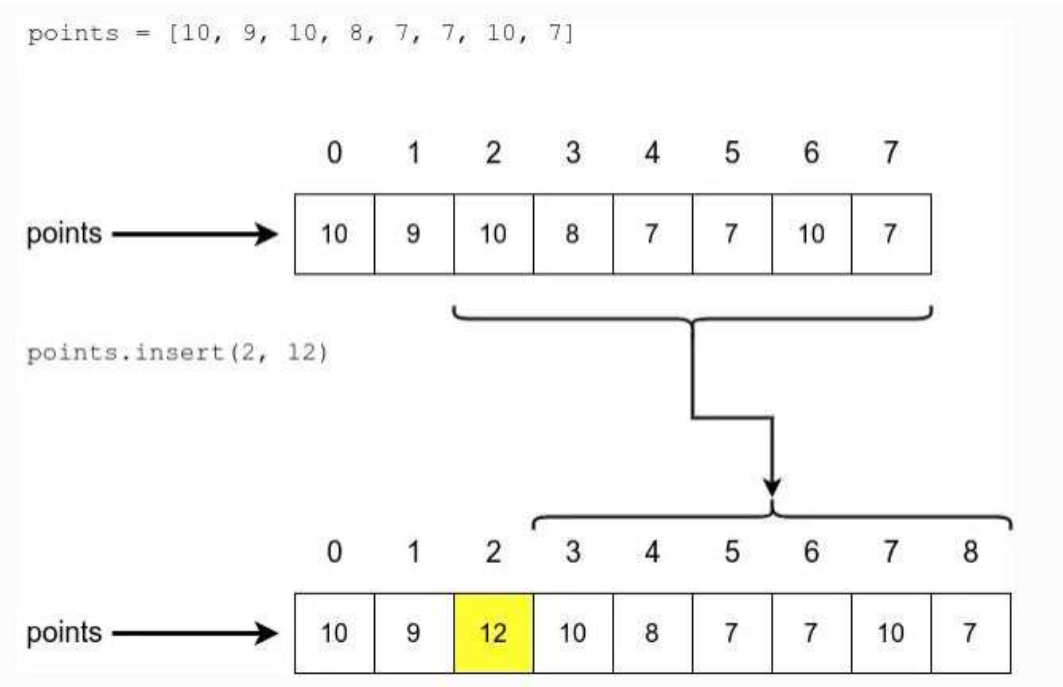
Items can be added with functions **append** and **insert**

`list.append(item)` appends at the end of the list

`list.insert(index, item)` inserts into given index

Adding items (2)

If item is inserted into middle, all following items are moved one step forward



Removing items

Method **pop** removes and returns an item at given index

Method **remove** removes item with given value - if more than one exist, only the first one is removed

Sorting lists

Method **sort** sorts the items "in place"

Function **sorted** returns a sorted copy of the list

Iteration

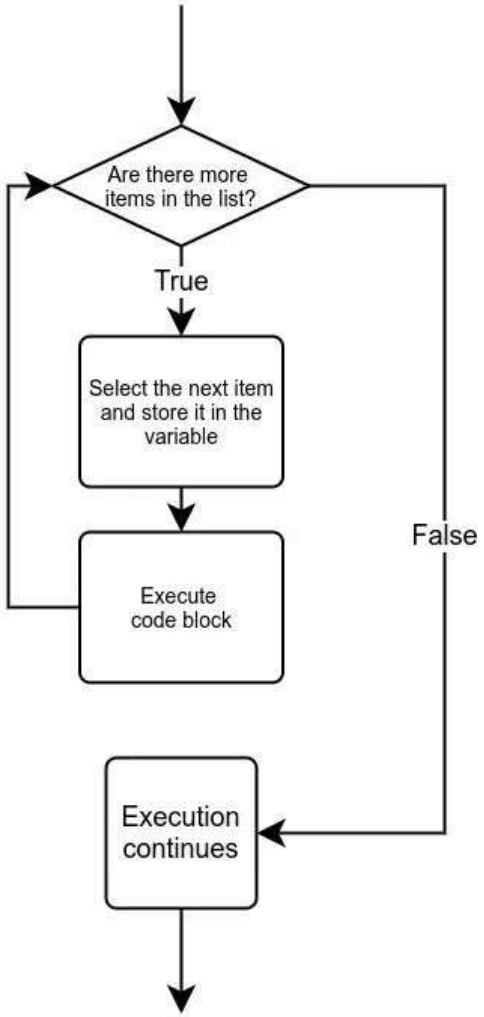
We have used the while statement to iterate the items before

```
my_list = [3, 2, 4, 5, 2]

index = 0
while index < len(my_list):
    print(my_list[index])
    index += 1
```

Iteration with for loop

for loop is a good choice when we want to iterate through all items in a sequence



Function range

Function **range** can be used to create an iterable sequence

This is handy if we want to iterate through a sequence, but do not need the list of values

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Slicing lists

Syntax is similar to substrings

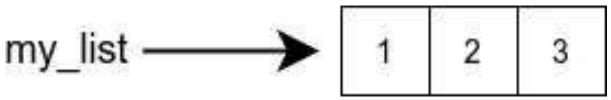
```
list[start : end]
```

or

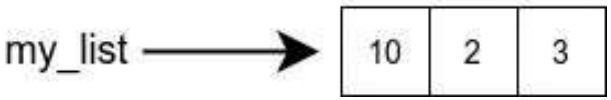
```
list[start : end : step]
```

Immutability of strings

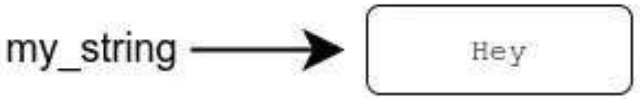
```
my_list = [1,2,3]
```



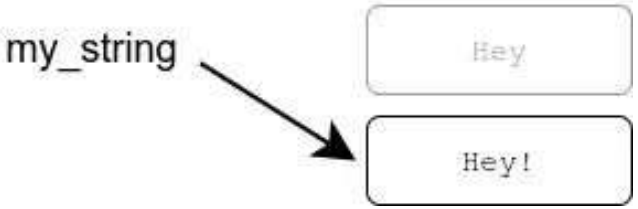

```
my_list[0] = 10
```




```
my_string = "Hey"
```




```
my_string = my_string + "!"
```



Some useful methods

Method **count** returns the number of items in a sequence

Method **replace** creates a new string with given substring replaced with another substring

Formatting output

Concatenation:

```
name = "Mark"
age = 37
print("Hi " + name + " your age is " + str(age) + " years" )
```

Separating with commas:

```
print("Hi", name, "your age is", age, "years" )
```

```
print("Hi", name, "your age is", age, "years", sep="")
```

f-strings

```
name = "Erkki"
age = 39
print(f"Hi {name} your age is {age} years")
```

```
number = 1/3
print(f"The number is {number}")
```

Next week

More lists

Multidimensional lists

List references

Dictionary

Tuple