

# KOLT Python

## Functions

İpek Köprülülü

Monday 11<sup>th</sup> March, 2019

# KOLT

1. Recap  
○○○○○○○○○○

2. Functions  
○

# Agenda

1. Recap

2. Functions



# Lists

- Group values together. `my_values = [1, 'a', None]`
- You can think of each element as a variable, accessed by **indexing**
- You can do everything you do to variables to list elements:
  - Assign new values: `my_values[0] = 3`
  - Use shorthand assignment operators: `my_values[1] += 'bc'`
  - Learn their type: `type(my_values[2]) # => <class 'NoneType'>`
  - Change their type: `my_values[2] = True`
  - Compare their value: `if my_values[0] == my_values[1]: ...`
- What happens when we call `my_values[3] = 3? # => IndexError`

## List Indexing

Access elements at a particular index

0	1	2	3	4
['a',	'b',	'c',	'd',	'e']
-5	-4	-3	-2	-1

```
x = [1, 2, 'a', 'hello']  
x[0] # => 1  
x[1] # => 2  
x[2] # => 'a'  
x[3] # => 'hello'  
x[-1] # => 'hello'  
x[-2] # => 'a'  
x[-3] # => 2
```

## List Slicing

Access collection of elements by specifying **[start:stop:step]**  
Gives a list, even when number of elements is not bigger than 1.

```
numbers = [0, 1, 2, 3, 4, 5]
"""
ASCII art analogy :)
+---+---+---+---+---+---+
| 0 | 1 | 2 | 3 | 4 | 5 | => Indices
+---+---+---+---+---+---+
0   1   2   3   4   5   6 => Borders
-6  -5  -4  -3  -2  -1
"""
```

```
numbers[0::2] # => [0, 2, 4]
numbers[:] # => [0, 1, 2, 3, 4, 5]
numbers[1:] # => [1, 2, 3, 4, 5]
numbers[-2:] # => [4, 5]
numbers[1:4] # => [1, 2, 3]
numbers[1:1] # => []
numbers[-99:99] # => [0, 1, 2, 3, 4, 5]
numbers[::-1] # => [5, 4, 3, 2, 1, 0]
numbers[::-2] # => [5, 3, 1]
```

Slices with `step = 1` are called **Basic Slice**.

Slices with `step != 1` are called **Extended Slice**.

## List Mutation

`list.append(x)`: Append x to end of the sequence

`list.insert(i, x)`: Insert x to index i

`list.pop(i=-1)`: Remove and return element at index i

`list.remove(x)`: Remove first occurrence of x

`list.extend(iterable)`: Add all elements in iterable to end of list

`list[i] = new_value`: Update value of index i with new value

`list[basic_slice] = iterable`: Change elements in basic slice with elements in iterable, sizes can be different: `numbers[:]` = []

`list[extended_slice] = iterable`: Change elements in extended slice with elements in iterable 1-1, sizes must be equal.

## Some Other List Operations

**in** operator: Check whether an element is in list. `3 in numbers`  $\Rightarrow$  True

**len(list)**: Returns the length of list(and other collections).

**list.index(value, start=0, stop=len(list))**: Return first index of value.

**list.count(value)**: Count number of occurrences of value in list.

**list.reverse()**: Reverse the list (in-place)

**list.sort()**: Sort list elements (in-place)

For more, type `help(list)` in your interactive interpreter.

## Strings

Special kind of **lists**! `name = 'Ahmet'`

You can do:

- Indexing: `name[2] ⇒ 'm'`
- Slicing: `name[::-1] ⇒ 'temhA'`
- Search by `in` operator: `'hm' in name ⇒ True`

You can not do:

- String mutation: `name[2]='H' ⇒ TypeError`

Special functions about strings: `str.isnumeric()`,  
`str.capitalize()`, `str.format(...)`, `str.find()` ...



# Loops

Do something for many elements or based on a condition.

```
while <condition>:  
    <expression>  
    <expression>  
    ...
```

Similar to simple if blocks, but runs again and again until condition check fails.

```
for <item> in <iterable>:  
    <expression>  
    <expression>  
    ...
```

Iterable: collection of **ordered** elements.  
What is next after this item?

## For Loops

What is next after this item?

numbers[1] is after numbers[0]  $\neq$  **numbers[1]** > **numbers[0]**

**Examples of iterables:** lists, strings, ranges

## Ranges

`range(start, stop, step)`: creates a sequence of integers from start (inclusive) to stop (exclusive) by step.

Can be **indexed** and **sliced**

`len()` and `in` operator can be used

# For Loops

```
names = [Mario, Peter, Anna ,Paul ,Anna]

for number in range(2, 5):
    # In every iteration, we a have a different value from iterable
    # We can access the value with the name we specified
    print(number)
    # range is collection of integers, we can use ints in indexing
    print('Hello {}'.format(names[number]))

# Nested loops
for name in names:
    # In every iteration name changes, in the order of names
    if name != names[number]:
        print('{} says hello to {}'.format(name, names[number]))
```

## Break, Continue

**Break** terminates the closest for or while loop

```
for i in range(0, 5):  
    if i % 2 == 1:  
        break  
    print(i)
```

```
x = 1  
while x < 100:  
    x *= 2  
    if (x+1) % 3 == 0:  
        break  
    print(x)
```

**Continue** continues with the next iteration of the loop

```
for i in range(0, 5):  
    if i % 2 == 1:  
        continue  
    print(i)
```

```
x = 1  
while x < 100:  
    x *= 2  
    if (x+1) % 3 == 0:  
        continue  
    print(x)
```

## For Else, While Else

else in branching: executed when all of the conditions in upper if/elif blocks are False

else in loops: executed when loop is terminated **without** a break statement

```
while <condition>:
    <expression>
    if <condition>:
        break
    <expression>
    ...
# This block is executed if
# while loop is not terminated by break
# Note: this block runs even when
# condition is False at initial evaluation
else:
    <expression>
    <expression>
```

```
for item in iterable:
    <expression>
    if <condition>:
        break
    <expression>
    ...
# This block is executed if
# for loop is not terminated by break,
# it iterated all elements
else:
    <expression>
    <expression>
```

## 1. Recap

○○○○○○○○○○

## 2. Functions

