1 Cycles

1.1 FOR cycle

Basic iteration – repeat block of code for every element in a sequence.

We can iterate over a string:

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
for letter in alphabet:
    # variable letter is declared to be one character of alphabet in every
        ↪ iteration
    # number of iterations is equal to len(alphabet)
    print letter
```

Cycle through a list:

```
invitedPeople = ["Bob", "Alan", "Emily", "Gustav"]
for person in invitedPeople:
    print "Send invitation to", person

## This will print
# 'Send invitation letter to Bob'
# 'Send invitation letter to Alan'
# 'Send invitation letter to Emily'
# 'Send invitation letter to Gustav'
```

Use **range()** to create lists with incrementing numbers:

```
range(5) # → [0, 1, 2, 3, 4]
range(2,8) # → [2, 3, 4, 5, 6, 7]
range(1,18,3) # → [1, 4, 7, 10, 13, 16]
```

Useful for getting not only the value, but its index too:

```
for position in range(len(cities)):
    print "City no.", position + 1, cities[position]
# will print "City no. 1 Praha", "City no. 2 Brno"
```

Conditions inside cycles:

```
polygons = [
[[1,7], [1,3], [2,3], [2,7]], # polygon 1
[[1,1], [1,5], [3,3]], # polygon 2
[[0,0], [0,5], [2,10], [4,7], [3,10], [8,2]] # polygon 3
]

for polygon in polygons:
    if len(polygon) < 3:
        print "This is not a polygon!"
    elif len(polygon) == 3:
        print "This is a triangle"
    elif len(polygon) == 4:
        print "This has four sides (might be a square?)."
    else:
        print "This is a more complex polygon."
```

Creating a line with random coordinates:

```
import random # Loading some library for generating random numbers

length = 10 # for example
length = int(raw_input("How long should the line be? ")) # or we can ask
    ↪ the user!

line = []
for i in range(length):
    coordinates = [random.random() * 1000, random.random() * 1000]
    line.append(coordinates)

print line
```

1.2 While

The **while loop** will run until a specified **condition is met**:

```
count = 5
while count < 5:
    count += 30
    print "Count has changed to: " + str(count)

print "Count is now bigger than 100, it is: " + str(count)
```

```
# Simulating drawing cards in blackjack
total = 0
while total < 17:
    total += 2 * random.randint(1,5)

if total > 21:
    print "The dealer lost!"
else:
    print "The dealer has " + str(total) + " points."
```

Be very careful about infinite loops:

```
i = 5
while i < 6:
    print i
```

Note

Another way to break a while loop is to make the condition always true and break it when some condition is met:

```
while True:
    # run some code
    if some_condition:
        break
```

2 Dictionaries

Another data type useful for storing data is a **dictionary**. We can iterate over it (same as with lists), but we **define** our own **keys** for the stored items:

```
# great way to store coordinates
coords = { "lat": 49.1876, "lon": 16.3273, "elev": 420.3 }

# or attributes
row = {
    "id": 0,
    "city": True,
    "name": "Brno",
    "population": 377440,
    "ranking": 2,
    "latitude": 49.2,
    "longitude": 16.6,
    "universities": ["MUNI", "VUT", "MENDELU", "VFU", "JAMU", "UNOB"]
}
```

Items are accessed similarly to lists, but using our own keys:

```
print coords["lat"]
print row["population"]

# if we don't have numbers as keys, this will not work:
print row[0]

# but this will:
students = {
    1: "Kristýna",
    2: "Petr"
}
print students[1]
# again, this will not:
print students[0]
```

Values can be of any type (integers, floats, strings, lists, dictionaries).
Some methods and operations for dictionaries:

```
a_dict = {'a': 156, 'b': 89, 'c': 41, 'd': 547}
print a_dict.items() # dict_items([('b', 89), ('a', 156), ('d', 547), ('c',
    ↪  41)])
print a_dict.keys() # dict_keys(['b', 'a', 'd', 'c'])
# list(a_dict.keys()) # returns ['b', 'a', 'd', 'c']
print a_dict.values() # dict_values([89, 156, 547, 41])
# list(a_dict.values()) # returns [89, 156, 547, 41]
print len(a_dict) # 4

# get() returns None if key is not defined and not an error
print a_dict.get(4) # None
print a_dict.get('a') # 156
```

Why should you use dictionaries?

- switch-case (alternative to conditions):

```
time = int(input('what hour is it?')) # 10

what_to_do = {
    5: 'you should be sleeping',
    8: 'make a breakfast',
    10: 'have a coffee',
    15: 'go to shop',
    20: 'have a shower'
}

print what_to_do[time] # have a coffee
```

- advanced constructions (simple alternative to databases):

```
countriesStats = {
    'Nigeria' : {
        'GDP': 1109000,
        'rank': 20,
        'languages': ['English']
    },
    'South Africa' : {
        'GDP': 725004,
        'rank': 30,
        'languages': ['Zulu', 'Xhosa', 'Afrikaans', 'English']
    },
    'Ethiopia' : {
        'GDP': 132000,
        'rank': 65,
        'languages': ['Amharic']
    }
}

print countriesStats['Nigeria']['languages'] # ['English']
```

- Another examples are file formats **JSON** and **GeoJSON** (more on that the next time)