2. DERS 10.06.2021

* stack overflow ⇒ İnsanlar birbirlerine soru-cevap yapıyorlar. Burdan faydalanabilirsin sorularına

* 79 karakteri geçmemeye çalış.

```
print(_____)
```

79 en fazla.

* Fazladan boşluk koymana gerek yok. Normal satır boşlukları yeterli.

df[0,] → yanlış

* =, +=, ==, <, >, >=, in, not in, is, is not, and, or, not

Bunlardan önce ve sonra birer boşluk bırakılabilir

* # Comment line (yorum satırı)

↳ ctrl+alt+3

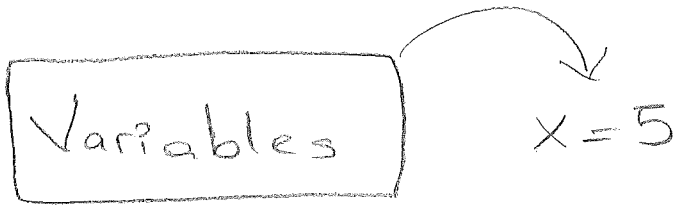
* Commentten önce 2 boşluk bırak.

```
print("Safa")
# ___
# ___
# ___
```

} Çoklu yazabilirsiniz. (Multi comment)

Table of Contents

* Variables (Değişkenin içine değer atılır.)
* Introduction to Data Types
* Strings
* Numeric Types
* Boolean
*



Yazarken PEP 8'e ve Python kurallarına uyacak.
İsimlendirmeler anlamlı olmalı.

variable name = variable

Assign a value to a variable

print = 'Jupyter'
price = 140
pi_number = 3.14

(1) FIRST LESSON 10.06.2021

What is IT ?

Kullanılacak programlar
- Python Playground
- Visual Studio Code
- Jupyter Notebooks
- Google Colab (Bilk bu)

→ Belli bir seviyeden sonra bu

Colab ⇒ Google. my drive'a kaydediyor

- Playground

Shift + Enter ⇒ Kod çalıştırma

print ('Hello World!') → String

print ("Hello World!") → String

Enter yapamazsın.

Triple quotes ⇒ print (""" Hello """)

(İçinde tekli veya ikili tırnak kullanılabilsin diye garantiye alır.)

print (""" Hello """)

Daha çok uzun metinlerde kullanılır. Enter'layıp alta geçebilirsin. Ama sıfır ve tekli tırnakta bunu yapamayız.

* Ondalık sayılar

print('3.14') → string tip

print (3.14)

* print (""" We should have enough time for family """)

Üçlü tırnak ve karşısında onu bitiren 3'lü tırnağı gördüğü için aradakileri kabul eder

* Empty line ⇒ print() or print(' ')

print ('Safa')
print () ——→ Burayı atlar
print ("""Acar""")

! print (' ') → Hata vermez.

print (''') → Python ilk önce 3'lüyü ele alır. O yüzden hata verir.

print ('''''') → Hata vermez

* PEP 8 ⇒ Kütüphane
Python'un görgü kuralları
Adab-ı muaşeret
Anayasa

12 — 14 Haziran

Strings

* Her zaman tırnak içine yazılır. Stringin içine herşey yazılabilir.

```
my_text = 'Hello'
print(my_text)
```

* Strings are identified as a set of characters represented in the single or double quotes.

"632"
"It's okay!"

} Print yazmadan da çıktı alır. (Jupiter'de) Tırnak içinde yazılan sayılar stringtir.

```
in: type(my_text)
out: str
```

NUMERIC TYPES

Three basic numeric types in Python:

* Integers
* Floats
* Complexes (a+bj, x+yj)

1) INTEGER

Tamsayılar (Whole numbers) → interviewde çıkar.

```
my_number = 40
negative_num = -18
print(my_number)
```

} Atamak = Assign etmek

Variable name = Value

(2) FLOAT → real numbers with a decimal point

*Noktalı sayılar. (Ondalık sayılar)

```
my_float = 40.0
negative_float = -18.66
print(my_float)
print(negative_float)
```

(3) BOOLEAN

*True or false. Başka değer almaz. (Baş harfler büyük)
*Boolean type is called bool.

TYPE CONVERSION ⇒ Bir tipten başka bir tipe geçiş.

*The value of any type in Python can be converted (dönüştürme)
to a str

```
str()
float()
```

Converting float to str

```
P = 3.14
converted_pi = str(pi)
print(converted_pi)
print(type(converted_pi))
```

stringe dönüştürdük.

out ⇒ '3.14'

(3) 14 Haziran

Converting float to int

```
pi= 3.14
converted_pi = int(pi)        → int'e çevirdik
print(converted_pi)
print(type(converted_pi)
```

out : 3 ⟶ Sadece tam kısmı alır.
int

Converting int to float

```
no=3
converted_no=float
print(converted_no)
print(type(converted_no)
```

out : 3.0
float

"12" +12 ⇒ Error (İki farklı tip bir str diğeri int)

"12" +"12" ⇒ 1212 (string)

12 +12 ⇒ 24 (integer)

TABLE OF CONTENTS

ARITHMETIC OPERATIONS

// → Bölmeden sonra tam kısmı alır.

*Addition operator ⇒ +

*Substraction ⇒ -

Multiplication ⇒ *

Float Division ⇒ / (Bölme işleminde sonuç her zaman FLOAT)

Integer Division ⇒ //

Exponentiation ⇒ ** ("Üs alma, Karekök alma)

Remainder ⇒ % Kalan : 10 % 3 = 1

print ('4' + 4) ⇒ ERROR

x = 5
x = x + 2 veya y += 2

r = 5
area = ?

pi = 3.14
r = 5
area = pi * (5 ** 2) → Kare alma

area

out ⇒ 78.5

5

17.06.2021

OR ⇒ True ise ilk true
Sadece bunu ezber

Three boolean operators : and
or
not → ya true ya da false'dur.

STRING DATA STRUCTURES

1. Arithmetic Syntax (+, *, =)
2. % Operator formatting
3. string.format( ) method
4. f-string formatting

and ⇒ Hepsi true en sondaki
1 tane bile false ilk false

or ⇒ Hepsi false en sondaki
1 tane bile true ilk true

ARITHMETIC SYNTAX

```
print ('no' + 'way')
```
output → noway

```
print (3*"no way!")
```
output → no way!no way!no way!

```
fruit = 'orange'
fruit += 'apple'
print (fruit)
```
output → orangeapple

% OPERATOR

'%s' → stands for 'string'

'%2f' → stands for 'float'

'%d' → stands for 'numeric'

```
phrase = 'I have %d %s and %2f brothers' % (4,'children', 5)
print (phrase)
```
output ⇒ I have 4 children and 5.00 brothers

STRING.FORMAT()

It is the improved form of % operator formatting.

```
fruit = "orange"
vegetable = "tomato
amount = 4
print ('The amount of {} and {} we bought are {} pounds'
        .format (fruit, vegetable, amount)
```

output → The amount of orange and vegetable we bought are 4 pounds.

F-STRING

It makes string formatting easier.

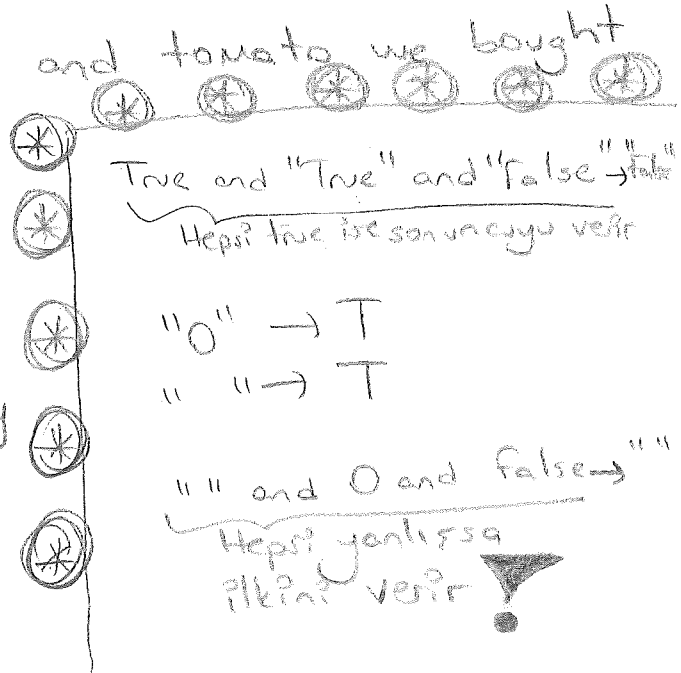
```
fruit = 'orange'
vegetable = 'tomato'
amount = 6
result = f"The amount of {fruit} and {vegetable} we bought
          are totally {amount} pounds."
print (result)
```

output → The amount of orange and tomato we bought are totally 6 pounds.

FALSY

'none
0
0.0
{}
[]
" "

Bunların dışında hersey truly 'dir.



(6)

17.06.2021

$X += 3 \Rightarrow x = x + 3$

$X *= 3 \Rightarrow x = x * 3$

$x **= 3 \Rightarrow x = x ** 3$

Orders

1. parentheses : ()
2. power :
3. unary minus :
4. multiplication and division : *, /
5. addition and subtraction : +, −

(*) sep ⟶ space

end ⟶ newline

sep = ' ' → Arasına ne koyayım?

end = '\n' → Sonuna koy

default : varsayılan değer

(*) print ("ali\nveli\ndeli") # Arada boşluk yok.

```
ali
veli
deli
```

(*) print (12, "22", False) # Virgül aralara boşluk koyar.

12 22 false

(*) print (12, "22", false, sep = "-") → # Araya bundan koy demek

12-22-false

(*) print("hayriye", end = "-")
print("hayri")

hayriye-hayri

İçindekileri yazdırdıktan sonra araya birşey koymaz. Yan aşağı iner, koyarsan onu yapar.

(*) \n ⇒ new line
\t ⇒ tab
\b ⇒ backspace

} Bunlar tırnak içinde kullanılır. Bunlar string'tir.

→ sondaki harfi sil demek.

*print('it\'s essential')
it's essential

Tek tırnak kullandım, içerde tekrar kullanamam o yüzden içerdeki (') nın etkisini kaldırmak için (\) koyuyoruz.

BOOLEAN

(*) print('4'+4)
Type error (İki tip bir arada olmaz)

ORDER

1. not
2. and
3. or

} Önce not, sonra and, sonra or işlenir.

(*) print(none or 1)

1

print([] or 1)

1

print({} and 1)

0

print("" or "!!!") ⇒

"" → False
"!!!" → True

a = "" → False demek
b = " " → True demek
c = "False" → True demek
d = True → True demek

7 | 21.06.2021

str.strip() ⇒ Her iki taraftaki tüm boşlukları kaldırır. (veya belirtilen karakterleri)

str.rstrip() ⇒ sağ taraftaki boşlukları kaldırır. (veya belirtilen karakterleri)

str.lstrip() ⇒ sol taraftaki boşlukları kaldırır. (veya belirtilen karakterleri)

* "%s" ⇒ String

"%d" ⇒ Numeric

"%0.2f" ⇒ float

\b ⇒ backspace

\n ⇒ new line

\\ ⇒ back slash

\' ⇒ single quote (')

\" ⇒ double quote (")

\t ⇒ tab

end= ⇒ specify the end of the string that we print

(8)

✳ What are the string.startswith() and string.endswith() method used for? Describe how?

To search patterns in a string there are two useful methods called startswith() and endswith() that search for the particular pattern in the immediate beginning or end of a string and return TRUE if the expression is found.

UPPER CASE : Büyük harf
LOWER CASE : Küçük harf

✳ str.swapcase() ⟹ Büyük harfi küçüğe, küçüğü büyüğe çevirir

str.capitalize() ⟹ İlk karakter büyük, diğerleri küçük

str.upper() ⟹ Hepsi büyük harfe dönüşür

str.lower() ⟹ Hepsi küçük harfe dönüşür

str.title() ⟹ Her kelimenin ilk harfi büyük

str.count() ⟹ Parantezin içine yazdığın harfin kaç tane olduğunu söyler.

```
safi = "I am a student"
title_safi = safi.title()
print(title_safi)
lower_safi = safi.lower()
print(lower_safi)
upper_safi = safi.upper()
print(upper_safi)
capitalize_safi = safi.capitalize()
print(capitalize_safi)
```

str.replace(";","I") ⟹ ; yerine I koyar.

9 | 21.06.2021

Kadayıf Dolması

*Kadayıf için için
* Fındık
* Fıstık
* Ceviz

Şerbet için
* Su
* Şeker

Yumurta

```
kadayıf = True
# iç
fındık = True
fıstık = True
ceviz = True
# şerbet
su = True
şeker = True
yumurta = True
# içecek
çay = True
limonata = True
meşrubat = True
```

Olmazsa olmazları and ile birbirine ince bağla

yedim = kadayıf and iç and şerbet and yumurta and içecek

yedim = kadayıf and (fındık or fıstık or ceviz) and (su and şeker) and yumurta and (çay or limonata or meşrubat)

if'li olaylar, and ve or larla yapabiliriz.

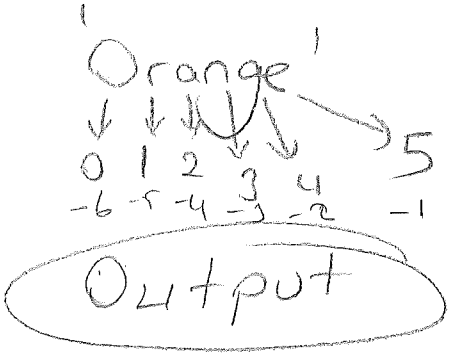
- print(" " ") → Error verir.
- print("\\" ") → \"
- print("\" ") → Error verir.

* String type, iterable veri tipidir.

ITERABLE: Elemanlarına ayrılabilen ve bu elemanlar alınıp tek tek kullanılabilir, döngüye sokulabilir.

```
fruit = 'Orange'
print('Word          :', fruit)
print('First letter  :', fruit[0])
print('second letter :', fruit[1])
print('3rd to 5th letters :', fruit[2:5])
print("Letter all after 3rd :", fruit[2:])
```

↳ Köşeli parantez



[start : stop : step]

↓ bu alınmıyor. bir öncesinde kalıyor.

Word : orange

first letter : O

Second letter : r

3rd to 5th letters : ang

Letter all after 3rd : ange

(10)

* isim = "clarusway okulu"

isim [3:7]

output ⇒ rusv

* orange

-6 -5 -4 -3 -2 -1
0 1 2 3 4 5

fruit[1:4] veya fruit[-5:-2]

[start : stop : step]

Başladığı yerden başla, bir öncekinde bitir.

* city = phoenix

0 1 2 3 4 5 6
-7 -6 -5 -4 -3 -2 -1

city[::] → Baştan başla sona git
phoenix

city[::-1] → Sondan başla başa git
xineohp

city[-3:] → -3'den başla sona kadar git
nix

Negatif index ile negatif step farklı şeyler.

city[-3::-1] → -3'den başla sona kadar git.
Son burda boş oluyor.
Çünkü step (-)
neohp

* Negatif stepte son deyince baştaki son. olur.

len → Uzunluk

```
vegetable = 'Tomato'
print ('lenght of the word', vegetable, 'is:', len(vegetable))
```

output : length of the word Tomato is : 6

STRING FORMATTING + =

* ~> Repeat yapar.

```
str_one = 'upper'
str_two = 'case'
str_comb = str_one + str_two
print ('upper' + 'case')
print (str_one + str_two)
print (str_comb)
```

} Hepsi de uppercase

(*) * ~> Repeat yapar

```
str_one = 'upper'
str_two = 3* 'upper'
str_comb = str_one * 3
print (str_two)
print (str_comb)
print (*str_one)
```

→ upperupperupper
→ upperupperupper
→ u p p e r

↳ yildiz en başta kullanilirsa boşluk birakir. Ama fiziki bir space değil. Sadece yapiyor bir nevi. Görsel yansitma.

11

* city = 'Chicago'
print(city[::-1]) → En sondan başa kadar git. başla en başa

ogacih C

+=

```
str_one = 'upper'
str_one += 'case'
print(str_one)
str_one += 'letter'
print(str_one)
str_one += 'end'
print(str_one)
```

} uppercaseletterend

```
str1 = str1 + str
str1 += str
```

```
str1 = str1 * 2
str1 *= 2
```

string.format() Method

'string {} string {} string'.format(data 1, data 2)

↳ ÖNEMLİ

↳ Bu nokta method anlamına gelir.

* "Benim adım {}." .format("Safiye")

→ Benim adım Safiye

tonguç

tonguç

longue

int 22 ⇒ Elemanlarına ayrılmaz.

str "22" ⇒ Karakterlerden oluşur. Elemanlarına ayrılabilir.

f-string ⇒ Elemanlarına ayrılabilir.

```
f'string {variable1} string {variable2} string'
```

f-format formül

(*) sample = f"{2**3}"
print(sample)
output ⇒ 8

} Süslü parantezde işlem yapmak mı?? OMG

(*) x=2
y=2
print(f"Burada neler oluyor? {x**y}")
output ⇒ Burada neler oluyor? 8

(*) name = 'MARIAM'
output = f"My name is {name.capitalize()}"

AYNI

Parantez kullanmadan kullanarak aynı şeyi yapar.

```
name = "Safi"
job = "teachers"
domain = "Data Science"
message = (
    f"Hi {name}."
    f"You are one of the {job}."
    f"in the {domain} section."
    )
print(message)
```

```
name = "Safi"
job = "teachers"
domain = "Data Science"
message = f"Hi {name}." \
    f"You are one of the {job}" \
    f"in the {domain} section."
```

MULTILINE

```
name = "Susan"
age = "young"
gender = "lady"
school = "CLRWY IT university"
```

"Susan is a young lady and she is a student at the CLRWY IT university."

```
message = f"{name} is a {age} {gender}" \
    f"she is a student at the {school}."
```

TABLE of CONTENTS

1) IMMUTABILITY → Değiştirilemez.

Örneğin stringler değiştirilemez. (Mantıken)
Listeler değiştirilebilir.

```
var_string = 'clarusway'
print(var_string.lower())
print(var_string)
var_string = 'clarusway'.lower()
print(var_string)
```

We can't change the string

→ Tekrardan tanımlamak gerekiyor.

To change string, we should reassign the new (changed) string value to a variable

output : clarusway
Clarusway
clarusway

Değiştirmek için ikinci bir adım gerekiyor.

```
var_str = 'In God we Trust'
var_str.lower()
print(var_str)
```

output ⇒ In God We Trust

String'i değiştiremeyiz

```
var_str = var_str.lower()
```

output ⇒ in god we trust

Kalıcı değiştirmek için yeniden atıyoruz.

(2) SEARCHING A STRING

* string.startswith()

Starts searching from the beginning to the end.

* string.endswith()

Starts searching from the very end to to the beginning.

(*)
```
text = 'www.clarusway.com'
print(text.endswith('.com'))
print(text.startswith('http:'))
```

output ⇒
```
True
True
```

(3) CHANGING A STRING → Bir sürü method var.

string.method(arguments)

String bir veri tipi. Stringin sonuna nokta koyup üstüne bina edilen metodlar var.

(*) str.replace(old, new[, count])

→ Köşeli parantezler optional'dır. Yani yazsan da olur yazmasan da. Kaç tane yapacağını sor gibi.

str.swapcase() → Büyüğü küçük, küçüğü büyük yapar.

tonguc

Python library → Beginner kurslar var.

LMS → Python → Additional Resources (izle)

All these methods return `str` type. So we can use the following syntax.

```
string.method1().method2().method3
```

(4) EDITING A STRING

Sağını solunu kırp, düzenleme yap.

```
string.method(arguments)
```

→ Yazacağın birşey yoksa boş bırakabilirsin.

string.strip() ⇒ Bütün boşlukları kırp.

string.rstrip() ⇒ Sağdan boşlukları kırp.

string.lstrip() ⇒ Soldan boşlukları kırp.

İçlerinde birşey yazarsa örneğin

```
text = "Safi"
print(safi.strip("si"))
```

→ istersen "is de önemli değil

output: af

LIST

There are various collection types in Python. While types such as int and str hold a single value, collection types hold multiple values.

One of the most useful collections in Python is a list. In Python, a list is only an ordered collection of valid Python values.

In your programs, you usually need to group several items to render as a single object. We use collection types of data to do this job.

```
country = ['USA', 'Brasil', 'UK', 'Germany', 'Turkey']
print(country)
```

output: ['USA', 'Brasil', 'UK', 'Germany', 'Turkey']

```
string_1 = 'I quit smoking'
new_list_1 = list(string_1) # we created multi element list
print(new_list_1)

new_list_2 = [string_1] # this is a single element list
print(new_list_2)
```

output:

```
['I', ' ', 'q', 'u', 'i', 't', ' ', 's', 'm', 'o', 'k', 'i', 'n', 'g']
['I quit smoking']
```

```
warning = 'you must quit smoking'
print (len (list(smoking)))
```

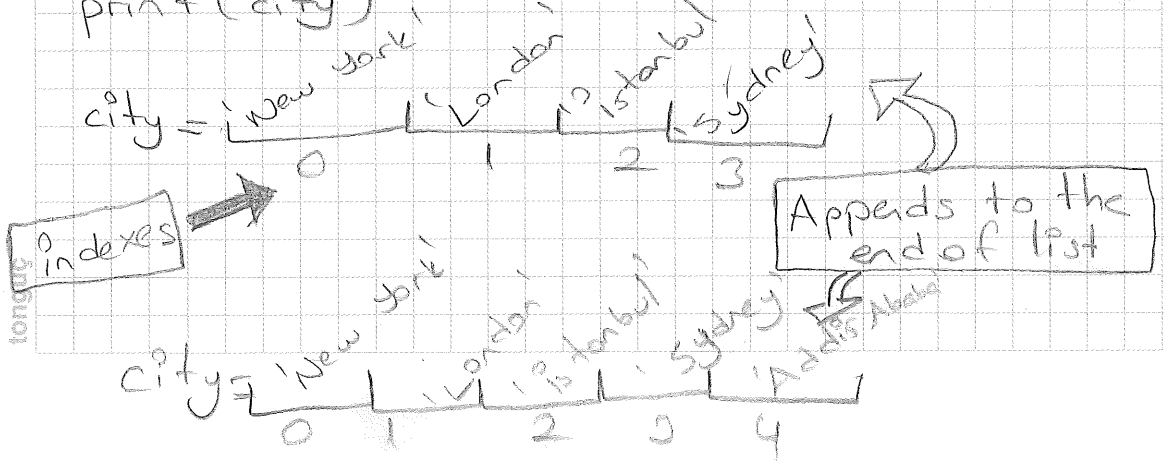
output ⇒ 21

.append() ➡ Append an object to end of a list. Using only list.append(element) syntax, return none. If you want to see the new appended list, you have to call or print it.

```
empty_list = []
empty_list.append ('1114')
empty_list.append ('plastic-free sea')
print(empty_list)
```

output ⇒ ['1114', 'plastic-free sea']

```
city = ['New York', 'London', 'Istanbul', 'Sydney']
city.append ('Addis Ababa')
print (city)
```



`insert()`: Add a new object to list at a specific index. The syntax looks like :

```
list.insert(index, object)
```

```
city = ['New York', 'London', 'Istanbul', 'Sydney']
city.insert(2, 'Stockholm')
```

output → ['New York', 'London', 'Stockholm', 'Ist.', 'Sydney']

```
list.remove()
```

```
city = ['New York', 'London', 'Istanbul', 'Sydney']
city.remove('London')
print(city)
```

output → ['New York', 'Istanbul', 'Sydney']

```
list.sort()
```
→ Alphabetical order

```
city = ['New York', 'London', 'Istanbul', 'Sydney']
city.sort()
print(city)
```

output → ['Istanbul', 'London', 'New York', 'Sydney']

index()
del()
pop()
search!

```
city = ['New York', 'Istanbul', 'London', 'Sydney']
print(len(city))
```

output ➡ 4

```
my_list = [1, 3, 5, 7]
print(my_list * 3)
```

output ➡ [1, 3, 5, 7, 1, 3, 5, 7, 1, 3, 5, 7]

```
city = ['New York', 'London', 'Istanbul', 'Sydney']
city[1] = 'Melbourne'
print(city)
```

output ➡ ['New York', 'Melbourne', 'London', 'Istanbul', 'Sydney']

Accessing List

You know that there are several types of collections for string data in Python like list, tuple, dictionary.

Each item or element in a list, as well as every character in a string, has an index corresponding to their location. Using indexes, we can access elements within a sequence. Now, let's see how can we do that?

```
colors = ['purple', 'blue', 'yellow', 'green']
print(colors[2])
```

output ⇒ yellow

```
city = ['New York', 'London', 'Istanbul', 'Sydney']
city_list = []
city_list.append(city)
print(city_list)
```

output ⇒ [['New York', 'London', 'Istanbul', 'Sydney']]

↳ double square brackets

elemanlarına ayrılabilen → iterable

```
city_list = [['New York', 'London', 'Istanbul', 'Sydney']]
print(city_list[0])
```

output → ['New York', 'London', 'Istanbul', 'Sydney']

```
print(city_list[0][2])
```

output → Istanbul

```
print(city_list[0][2][3])
```

output → a

Anlayamamıştın

```
fruits = ['apple', 'banana', 'watermelon', 'orange']
fruit_list = [] → Ekler
fruit_list.insert(0, fruits) → En başa fruits ekler
print(fruit_list[0][2][7])
```

output → l

```
fruit_list[0][2][7]
```

→ Liste içinde liste olduğu için. 0'ıncı eleman içteki listenin tamamı

→ İçteki listenin 2. elemanı ki bu da 'watermelon'

.insert ile fruit_list'in içine fruits listesini koyduk. Yani [['apple', 'banana', 'watermelon', 'orange']]

watermelon'un 7. indexi ki bu da 1 harf

COLLECTION TYPES

Collection fonksiyonlar iterable alır. Yani elemanlarına ayrılabilir. Stringler iterable dır.

1. List()
2. Set()
3. Tuple()
4. Dict()

MUTABLE (Değişebilirler)

LIST → String olmayanlar listelenemez!

Listeleri iki şekilde yaparız:

- [ ]
- list()

```
list_1 = ['h', 'a', 'p', 'p', 'y']
word = 'happy'
list_2 = list(word)
print(list_1)
print(list_2)
```

output ⇒ ['h', 'a', 'p', 'p', 'y']
['h', 'a', 'p', 'p', 'y']

```
list_3 = [word]
print(list_3)
```

output ⇒ ['happy']

The components of a list are not limited to a single data type.

Here is an example:

```
mixed_list = [11, 'Joseph', false, 3.14, None, [1,2,3]]
```

int, str, bool, float, NoneType, List

Liste içinde liste de olabilir.

```
X = [[1,2,3], [4,5,6]]
len(x) => 2
```

Liste içinde 2 liste var onları birer eleman olarak kabul eder.

(*) Create a list from string value of "2020's hard". Use both list() function and square brackets [].

```
my_list1 = ["2020's hard"]
my_list2 = list("2020's hard")
print(my_list1)
print(my_list2)
```

```
["2020's hard"]
['2', '0', '2', '0', ' ', 's', ' ', 'h', 'a', 'r', 'd']
```

0'ıncı elemanı

Sıfırıncı elemanın uzunluğunu sorgular.

```
print(len([[1,2,3][0]))
```

3

```
print(len([[]]))
```

1

BASIC OPERATIONS with LIST

How to create empty list? → `[]` `list()`

Boş liste `.append()` `.insert()` ile doldurulur.

```
numbers = [1,4,7]
numbers.append(9)
numbers.append('g')
print(numbers)
```

output ➡ `[1,4,7,9,'g']`

`.append()` ➡ Appends an object to the end of a list.

Create an empty list and then collect the int numbers(1-4) one by one into the list you created using .append() method.

```
sayılar = []
sayılar.append(1)
sayılar.append(2)
sayılar.append(3)
sayılar.append(4)
print(sayılar)
```

output ⇒ [1, 2, 3, 4]

`.insert()` ⇒ Araya girer. Yanındaki bi' omuz atar

```
numbers = [1, 4, 7]
numbers.insert(2, 9)
print(numbers)
```

(2 → index, 9 → number)

⇒ [1, 4, 9, 7]

```
numbers.insert(2, 6)
```

⇒ [1, 4, 6, 9, 7]

Create a list which consists of the int numbers (1, 2, 3, 4) and then insert number 5 at the end of the list using .insert() method

```
sayılar.insert(4, 5)
```

⇒ [1, 2, 3, 4, 5]

"".join(sorted(sentence))

.remove() ⟹ Eleman siler

```
numbers=[1,4,7,9]
numbers.remove(7)
print(numbers) ⟹ [1,4,7]
```

.sort() ⟹ Elemanları sıralar.
Ascii kodа göre sıralar!

```
numbers=[4,1,9,7]
numbers.sort()
print(numbers) ⟹ [1,4,7,9]
```

Slicing a List → Çıktı yine listedir.

```
numbers=[1,3,5,7,9,11,13,15,17]
print(numbers[2:5])
```

[5,7,9]

sequence[start:stop:step]

```
count=list(range(11))
print(count)
print(count[0:11:2]
```

[0,1,2,3,4,5,6,7,8,9,10]

[0,2,4,6,8,10]

- .append( ) → sona ekle
- .insert(index,number)→araya ekler. Yanındakine bir omuz atar
- .remove(number) → sil
- .sort( ) ⟶ Elemanları ASCII koda göre sırala.

Each part of the slice has a default (varsayılan) value, so they are optional. If we don't assign a value to the start index, it is considered to be 0; if we don't assign a value to the stop index, it will be the same as the length of the sequence.

my_list[:] ➡ returns the full copy of the sequence

my_list[start:] ➡ returns element from start to the end element

my_list[:stop] ➡ returns element from the 1st element to stop-1

my_list[::step] ➡ returns each element with a given step.

* numbers = ['1', '2', '3', '4', '5', '6', '7']

print(numbers[:])

['1', '2', '3', '4', '5', '6', '7']

print(numbers[3:]

['4', '5', '6', '7']

print(numbers[:5]

['1', '2', '3', '4', '5']

print(numbers[::2]

['1', '3', '5', '7']

* print(len[[12,34,56]][0]) ⊛ ?

3

Negative Indexing & Slicing

numbers = [1, 2, 3, 4, 5]

print(numbers[-4]

2

print(numbers[-3:]

3, 4, 5

print(numbers[:-3]

1, 2

print(numbers[::-1]

[5, 4, 3, 2, 1]

print(numbers[::-2]

[5, 3, 1]

```
odd_no = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
print(odd_no[7:3:-1])
[8, 7, 6, 5]
```
→ Tersten yaz ve 3. elemanı yazma

```
print(odd_no[2:6:-1])
[]
```
→ 2'den 6'ya tersten nasıl gideyim?

CURRENT LESSON

* len("a b")
  3
* len(["a b"])
  1
* len(["a b", False, None, []])
  4
  → Virgülle ayrıldığı için 4 sayıyor

→ Iterable liste (yani elemanlarına ayrılır

* len(1234) → int bir iterable değil!
  ERROR

Indexing a List

* print([1,3,6,9][2])

6

* saylar = [1,3,6,9]

print(saylar[2])

6

Aynı

* new = ["h", "a", 1, ["p", 2], True]

print(new[2])

1

print(type(new[2]))

int ⟶ int verir!

print(type(new[3]))

list ⟶ çünkü list te içinde

list
string
} iterable

* mix_list = [1, [1, "one", 2, "two", 3, "three"], 4]

(String numberları sec)

print(mix_list[1][1:6:2])

["one", "two", "three"]

* word = ['i', 's', 't', 'a', 'n', 'b', 'u', 'l']

print(word[2:-3])

['t', 'a', 'n'] → Sonuç verir mi? Verir, çünkü - + farketmez

print(word[-7:5])

['s', 't', 'a', 'n'] → Bu da olur. Her bir elemanın bir (+) bir (-) index numarası var. İstediğini kullan

print(word[-1:-6])

[ ] → Boş verir. Hata vermez.

print(word[-1:-6:-1] → Böyle sonuç verir. Çünkü geriye adım at diyorsun.

['l', 'u', 'b', 'n', 'a']

print(word[:4:-1] → Negatif başlan başlatıla göre git.

['l', 'u', 'b']

letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

print(letters[7:3:-1]

['h', 'g', 'f', 'e']

print(letters[2:6:-1] → 2'den 6'ya (-) yönde nasıl gideyim? Hiçbir index alamadığı için boş liste döndürür

[ ]

* `x = ([4, 6, 8])`
  `print(type(x))`
  `list`
  → Parantez içinde tek bir eleman var. Parantez almak yetmiyor. Bu bir liste

* `x = (5+3j)`
  `print(type(x))`
  `complex`
  → Tek elemanlı tuple yapamıyor Python

* `x = ([4, 6, 8],)`
  `print(type(x))`
  `tuple`

* `x = (5+3j,)` → Tek eleman oluşturmanın tek yolu bu tuple'da.
  `print(type(x))`
  `tuple`

Tuple List'den daha hızlıdır ve daha az yer kaplar.

TUPLES

*Bir collection type'tır. Yani birden fazla veriyi içinde barındırır.
*DEĞİŞTİRİLEMEZ! (Immutable)

List

- Daha yavaş. Çünkü değiştirilebilir. Bu ona bir ağırlık verir.
- Değiştirebilirsin.

Tuple

- Daha hızlı. Hafızada daha az yer kaplar.
- ID, password, TC no, applicationlarda kullanılır. Değiştirmek istemediğimiz şeylerde. Kazara değiştiremezsin.

TUPLE nasıl oluşturulur?

- ()
- tuple()

* ('happy') ⟹ String olur tuple olmaz. Virgül koyarsan tuple olur.

```
tuple_1 = ('h', 'a', 'p', 'p', 'y')
word = 'happy'
tuple_2 = tuple(word) → print(tuple_2)
('h', 'a', 'p', 'p', 'y')
```

tonguc

Tuple bir → Collection bir ⟶ Iterable

* a = tuple()
print(type(a))
tuple

* x = 1, 2, 3
print(type(x))
tuple

→ Parantez almadığın virgülle ayrılmış her bir iterable'ı Python tuple olarak algılar.

* y = 1,
print(type(y))
tuple

⟶ İstersen y = (1,) yap aynı şey.

* city = ["los-angeles", "beinit", "tokyo"]

city[0] = True
print(city)
[True, 'beinit', 'tokyo']

city[1:] = "istanbul", "seoul" → Bu bir tuple!
[True, 'istanbul', 'seoul']

city[1:] = ["1", "2", "3", 4, False] → (Liste)
print(city)
[True, '1', '2', '3', 4, False]

tonguç

```
city[1:] = "bangladesh"
print(city)
['True', 'b', 'a', 'n', 'g', 'l', 'a', 'd', 'e', 's']
```

▽ Tuple ve list birbirine çevrilebilir.

[1, 2, 3, 4] ⟶ (1, 2, 3, 4)

```
*mountain = tuple('Alps')
print(mountain)
('A', 'l', 'p', 's')
```

*Range fonksiyonu kullanarak 1'den 10'a kad tuple fonksiyonu yaz.

```
X = tuple(range(1, 11))
print(number, type(number), sep="\n")
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
<class 'tuple'>
```

shift+tab ⇒ kodun ne yaptığını gösterir

İndekslemeyi tuple'da da yapabiliriz.

(*) even_no = (0, 2, 4)
print(even_no[0])
0

(*) mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
↓ Six'e ulaş
0 1 2 3

print(mix_tuple[2])
[2, 'two', ('six', 6)] → Bunun tipi liste

print(mix_tuple[2][2][0])
'six'

mix_tuple[2].append("aktirma")
print(mix_tuple)
('11', 11, [2, 'two', ('six', 6), 'aktirmayın'], (5, 'fair'))

↓ Hani değiştiremezdim? Değiştiririm ama extra bir şey yapmak gerekir. Elemanlar değiştirilebilir.

DICTIONARY

{}
dict()

Value olarak
liste
boolean
string
int
} veri tipleri kullanılır.

* my_dict = {'key1':'value1'
'key2':'value2'
'key3':'value3'

key value
grocer1 = {'fruit':'apple', 'drink':'water'}
grocer2 = dict(fruit='apple', drink='water}
print(grocer1)
print(grocer2)

{'fruit':'apple', 'drink':'water'}
{'fruit':'apple', 'drink':'water'}

Key olarak liste olmaz tuple olur.
[ ] liste içine yazsam kabul etmez

* third_dict = {(1,2,3):"liste","clarus","the best"}
key bir tuple
string bir value

print(third_dict)
→ {(1,2,3):'liste','clarus':'the best'}

1. eleman
* fifth_dict = {"erkekler":{"ahmet":35, "mehmet":44,"cevat":19}, \
"bayanlar":{"selvi":22","bahar":55, "ayse":3}}
key value
2. eleman

tonguç

✳ state_capitals = {'Arkansas': 'Little Rock'
'Colorado': 'Denver'
'California': 'Sacramento'
}

state_capitals['Virginia'] = 'Richmond'

print(state_capitals)

→ Ensona 'Virginia': 'Richmond' ekler

✳ family = {"name1": "Safi",
"name2": "Bella",
"name3": "Ayse"}

print(family)
{'name1': 'Safi', 'name2': 'Bella', 'name3': 'Ayse'}

family["name4"] = "Tom"

family

→ {'name1': 'Safi', 'name2': 'Bella', 'name3': 'Ayse', 'name4': 'Tom'}

Dictionary'ler

Veri depolama (Bankalar müşterileri kategorize eder, Okullar öğrenci bilgilerini saklamada) kullanılır.

(Bu sefer parantez kullandık ()

* dict_by_dict = dict(animal = 'dog', planet = 'neptun', pi = 3.14, is_good = True)

print(dict_by_dict)

→ Bu dict fonksiyonunda () kullanıyorsak = kullanılır.

→ {'animal' : 'dog',
'planet' : 'neptun',
'pi' : 3.14,
'is_good' : True}

* cisimler = {"cisim1" : "mouse", "cisim2" : "havlu"}

print(cisimler)

→ {'cisim1' : 'mouse', 'cisim2' : 'havlu'}

MAIN OPERATIONS WITH DICTS

- .items()
- .keys()
- .values()

} Bunlar iterable. (Connection, for döngülerine sokabiliriz.)

tonguç

IN OPERATOR

İçinde var mı yok mu onu sorgular.

```
clarusway=["ali","veli","deli"]
"zeka küpü" in clarusway
→False
"deli" in clarusway
→True
```

NOT IN OPERATOR

```
"zeka küpü" not in clarusway
→ True
```

NESTED dicts

İç içe dictionary demek.

Keyler üzerinden gidilmez. Value'ler üzerinden gidilir.

(max (numbers, key=numbers.count))

ASSIGNMENTS (11.07.21)

1) numbers = [1, 3, 7, 4, 3, 0, 3, 6, 3]

- finds out the most frequent number in the given list
- Calculates its frequency
- Prints out the result such as

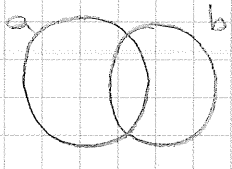
ANSWER

```
numbers = [1, 3, 7, 4, 3, 0, 3, 6, 9]
print ("numbers= ", numbers)
print (The most frequent number is in the list: ", (max (numbers, key=numbers.
                                                    count)
```

Her iki kümede ortak olan .intersection()

SETS (Matematikteki Kümeler) {} set()

Her eleman unik olursa, tekrar etmemek için bunlar kullanılır



- No repetition
- Math operations
  - Union
  - Intersection
  - Difference
- Unordered elements

{}
set() } Bu iki şekilde set yapılır.

→ Bunun içine giren eleman iterable olmalı

(*) set_1 = {'red', 'blue', 'pink', 'red'}

colors = 'red', 'blue', 'pink', 'red'

set_2 = set(colors)

print(set_1)

→ {'blue', 'pink', 'red'}

print(set_2)

→ {'red', blue', 'pink'}

(*) letter = " a b c d e f g h i j k l m n o p q r s t → arada boşluk

x w v z ".split()

print(letter)

→ ['a', 'b', 'c', ... 'v', 'z']

print(set(letter))

Setlerde sıralama yok.

→ {'v', 'k', 'x', 'm', 'w', 'd' ..... }

(*) test = {1, 'bir', 1.0}

len(test)

→ 2

(*) type({1})

dict

(*) type(set())

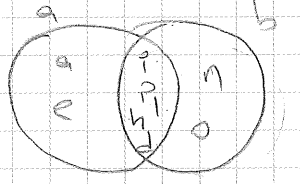
set

MAIN OPERATIONS with SETS

- add()
- remove()
- intersection()
- union()
- difference()

(*) a = set('phila delphia')

b = set('dolphin')



- Stringler iterible.

Her bir karakter tek tek alınır.

s(a∪b) = 9

s(a∩b) = 5

s(a-b) = 2

s(b-a) = 2

✱ a|b = a∪b ✱ ✱ a&b = a∩b ✱

print(a-b)
print(a.difference(b)) → Aynı sonuç

→ {'a', 'e'}

print(b-a)
print(b.difference(a)) → aynı

→ {'n', 'o'}

print(a|b)
print(a.union(b)) → aynı

Alt+kaldır imleç

→ {'p', 'h', 'i', ---- 'a', 'e'}

a.union(b) = b.union a

print(a&b)
print(a.intersection(b)) → aynı

→ {'i', 'p', 'l', 'd', 'h'}

✱ date = "12/07/2021"
set(date)
→ {"/", "0", "1", "2", "7"}

✱ b = {'09/01/2021'}
print(b)

→ {'09/01/2021'}

tongue

✳ given_list = [1, 2, 3, 3, 3, 3, 4, 4, 5, 5]

unique = set(given_list)

print(unique)

→ {1, 2, 3, 4, 5}

test = {given_list}

test

→ Type Error

ama şöyle yaparsan olur:

test = {1, 2, 3, 3, 3, 3, 4, 4, 5, 5}

test

→ {1, 2, 3, 4, 5}

✳ karışık = {[1,2,3], {1: "bir"}, ("iki", "üç")}

Type Error (Python 'collection'ları kümeye çeviremez.)

! Set'le boş küme yapılabilir.

TASK

Create two sets of string data from the capitals of the USA and New Zealand (eg: 'Madrid' → convert into a set)

- Perform all set operations.
  - Intersection
  - Union
  - Difference

```
usa_capt = set ('Washington')
nz_capt = set ('Wellington')
print (usa_capt)
print (nz_capt)
print (usa_capt - nz_capt)
print (usa_capt.difference (nz_capt))
print (nz_capt - usa_capt)
print (nz_capt.intersection (usa_capt))
```

→ syntax

* CONTROL FLOW STATEMENTS *

Programı kontrol etmeyi sağlayan, programa yön veren şey.

print("clarusway") +150 kere tekrar etmen gerek diyelim.
Yolu loops

if condition:
    body

A 4-space indentation

*if'ler true ve false ile çalışır. Buradaki condition değişkeni true-false olmalı.

(*) if "0":
    print("hello world")
→ hello world

(*) if 0:
    print("Hello world")
→ (Boş kalır. Çünkü bu false)

String

Python → Indentation
Case sensitive bir dil

None 0.0
[] 0j
() {}
"" false
0

→ Bunlar falsy, geri kalanlar truly.

(*) if True:
    print("it is true")
→ it is true

TASK

HAMBURGER

minced = True
bread = True
#green
lettuce = True
onion = True
grocer = True

hamburger = (minced and grocer and bread) and (lettuce or onion)
green

if hamburger:
    print("Bon Appetit!")
→ Bon Appetit!

Conditional statements ⇒ if
elif
else

* condition = (3>1) and (1<1)
if not condition:
print('It works')
else:
print('Something is wrong')

KAHOOT

* if a >= 22:

* price = '2200'
if price <= 2200:
print('I can afford it')
else:
print('It is expensive')

→ Type error

str
num

İkisi de str olsa yine de çalışır.

* "a" < "b"
→ True

* "4" < "39"
→ False (Str ilk harflere bakar 4<3 olarak algılar. Bu yüzden false)

tongue

COMPERASION OPERATORS

equal ==
not equal !=
greater than >
greater than or equal >=
less than <
less than or equal <=

Bunlardaki sonuç kesinlikle True-false ('in' 'not in' gibi) (and ve or da yine aynı true-false dönmekt.)

* empty_seat = 14
  if empty_seat > 3:
      print('There is still seat to sit') → True

→ 'There is still seat to sit'

* print(1 == 1)
  print("heavy" == "Heavy")
  print(12 < 12.1)
  print("hard" != "easy")

→ True
→ false
→ True
→ True

(*) string1 = set('TWELVE PLUS ONE')
string2 = set('ELEVEN PLUS TWO')
if string1 == string2:
    print('We are the same!')
else:
    print('We ere the not same!')

(*) a = input("Write yes or No:")
if "a": "yes"
    print("You entered true")

! IF KULLANMADAN ÇÖZ

convert = input("Enter yes or no: ").title().strip() == "yes"

Kullanıcı büyük-küçük kullanabilsin diye

Boşluk bırakson da sonra olmasın diye

Kullanıcıdan aldığım bir bilgiyi bool a çevirdik

print("You entered", convert)
→ You entered false

IF-ELSE STATEMENTS

if condition1:
    execute body1
else:
    execute body2

```
course = 'clarusway'
if course == 'clarusway':
    print ("You guaranteed the job")
else:
    print("Think about it again")
```

- if → Belirtilen koşul True ise altındaki kodu çalıştırır.

  elif → Önceki koşullar True değilse bu koşulu dene

  else → Önceki koşullar tarafından yakalanmayan her şeyi yakalar.

(*)
```
number = 5
if number <= 3:
    print ("Number is smaller than or equal to 3")
else:
    print("Number is bigger than 3")
```

TASK

Write a program to check whether a number entered by the user is even or odd. (Tek mi çift mi)

Print the result such as: "2 is even."

({}) yaparsam sıkıntı (5) olur.

```
a = int(input("Enter a number: "))
if num %2 == 0:
    print("Your number is an {} even.".format(a))
else:
    print (your number is a {} odd.".format(a))
```

TASK: Girdiğin sayı pozitifse pozitif, negatifse negatif desin.

```
a =float(input ("Enter a number"))
if a <0:
    print ("{} is a negative number".format(a))
else
    print ("{} is a positive number".format(a))
```

TASK: Write a program that prints which of the two numbers the user entered is large.
(Print the result such as: "The large number is 4")

```
num 1 = float (input ("Enter the first number: "))
num 2 = float (input ("Enter the second number: "))
if num 1 > num 2:
    larger = num 1
    else:
    larger = num 2
print("The larger number is:", larger)
```

2.yol

```
if num 1 > num 2:
print ("The larger number is:", num1
else:
print("The larger number is:", num2)
```

TASK

Convert boolean True to string value of "yes",
convert boolean false to string value of "No"

```
bool_value = False
if bool_value :
    print("yes")
else:
    print("No")
```

INTERVIEW

- if for the first one,
- elif for the rest, up until the final (optional)
- else for anything not caught by the other conditionals.

TASK

Kullanıcı 3 rakam girsin. En büyüğünü yazdır.

```
num1 = float(input("Enter first number: "))
num2 = "                    "
num3 = "                    "
if (num1 > num2) and (num1 > num3):
    largest = num1
```

```
elif (num2 > num1) and (num2 > num3):
    largest = num2
else:
    largest = num3
print("The largest number is:", largest)
```

TASK

Write Python Program to Check if a number is negative, positive or zero.

```
num = float(input("---"))
if num > 0:
    print
elif num == 0
    print
else:
    print
```

NESTED IF-ELIF-ELSE → Yuvalar of

```
audience_group = 'kid', 'teen', 'adult'
audience = "teen"
if audience in audience group:
    if audience == "kid":
        print("It is free go to cinema")
    elif audience == "teen":
        print(discounted price!")
    else: print("normal price")
else:
    print("No such audience, stay at your home!")
```

En distaki else

(*)
```
if x == 3 :
    print ("üç")
elif x == 4
    print ("dört"):
elif x == 5 :
    print ("beş")
else :
    print "(ənnuz")
    print ("Kod sonu geldi.")
```

TASK

Let's write a program that asks you to enter your exam score and calculates the range in which your degree is based on your exam score. The output would be: "your degree is B+"

95 and above → "A+"

90-94 → "A"

85-99 → "B+"

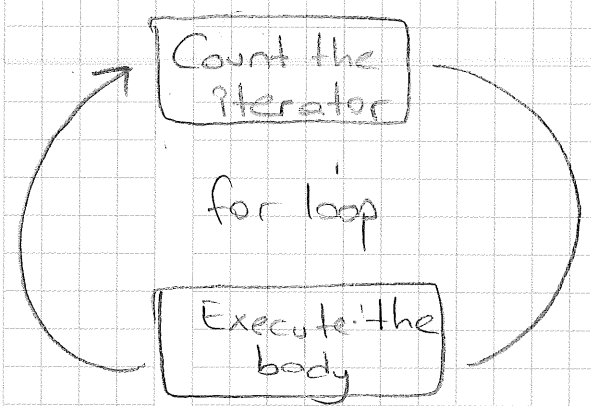
80-84 → "B"

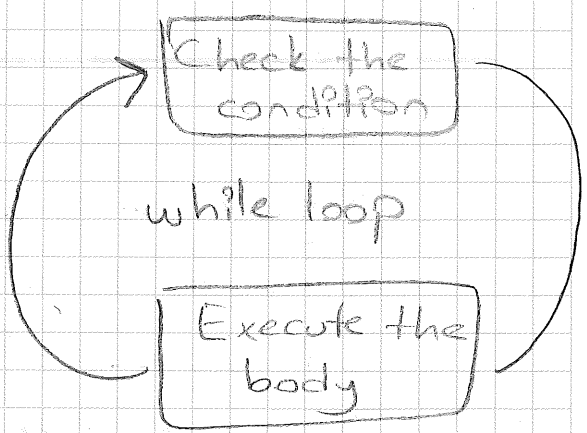
79 and below → "below B" or "B-"

```
score = float(input("Enter your score: "))
if score >= 90:
    if score >= 95:
        score_letter = "A+"
    else
        score_letter = "A"
elif score >= 80:
    if score >= 85:
        score_letter = "B+"
    else
        score_letter = "B"
else:
    score_letter = "B-"
print("your degree is: ", score_letter)
```

LOOPS

Can you explain the importance and logic of the loops?





KAHOOT

(*) condition = (3 > 1) and (1 < 1)

```
if 'condition':
    print ('It works')
else:
    print ('Something is wrong')
```

→ 'Something is wrong'

(*) condition = (3 > 1) and (-1 < 1)

```
if condition:
    print ('It works')
else:
    print ('Something is wrong')
```

→ 'It works'

→ str oldgu için kalıtım

(*) condition = (3>1) and (1<1)

```
if [condition']:
    print('It works')
else:
    print('Something is wrong')
```

Dolu bir liste oldugu için calisir.

→ 'It works'

(*) condition = (3>1) and (0<1)

```
if not condition:
    print('It works')
else:
    print('Something is wrong')
```

→ 'Something is wrong'

ASSIGNMENT ANSWER

numbers = [1, 3, 7, 4, 3, 0, 3, 6, 3]

→ The most frequent number is 3 and it was 4 times repeat.

MAX FONK.

* `max(1,2,3)`
→ 3

* `empty = []`
`max(empty, default='Safi')`
→ 'Safi'

* `seq = [1,1,1,1,2,2,2,3,3,4,4]`
`max(seq)`
→ 4
`seq.count(4)` # Kaç tane var?
→ 2
`seq.count(max(seq))` # En büyük kaç kez tekrar ediyor.
→ 2
`seq.count(1)`
→ 4
`max(seq, key=seq.count)` # Teker teker tüm elemanları saydırır, hafızada tutar. Max her zaman iterable'dan birini döndürür.
→ 1

↘ Saydırmak istiyorsan count'un içine sok

↘ # Sen elemanlarını tek tek say: Methodu uygula, ona göre seç.

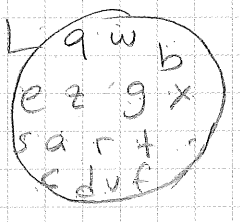
`seq.count(max(seq, key=seq.count))`

1'e uygular 4 çıkarır
2'ye uygular 3 çıkarır ....
Key'le ne yapacağına karar verir ve 4'ü seçer (Çünkü 4 kez tekrar etti)
4, 3, 2, 2 şeklinde bir iterable oluşur ve 4'ü seçer.

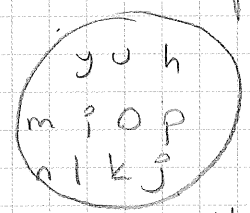
`print("En çok tekrar eden {} sayısıdır ve {} kere tekrar etmiştir.".format(a,b))`

ASSIGNMENT

Comfortable words (İki elle yazılan)



Sol elle yazılan



Sağ elle yazılan

test → a - L = boş (A'da olup L de olmayan yok)

a - r = a

Bu iki sonucu boolean ve and ile birbirine bağlıyoruz.

clonsway → Harfler her iki kümede de var. O yüzden True döner.

```
left = {'q', 'w', e, r, t, a, s, d, f, g, z, x, c, v, b}
right = {y, u, i, o, p, h, j, k, l, n, m}
word = "test"  (Örnek olarak)
set_word = set(word)  # Kelimeyi kümeye çevirdik
set_word
→ {'e', 's', 't'}
set_word - left
→ set()
set_word - wright
→
```

tonguç

left_bool = bool(set_word - left)

→ false

right_bool = bool(set_word - right)

→ True

tonguç

TASK

Find out if a given number is an "Armstrong Number"
An n - digit number that is the sum of the nth powers of its digits is called an n - Armstrong number. Examples:

1) $371 = 3^3 + 7^3 + 1^3$;
2) $9474 = 9^4 + 4^4 + 7^4 + 4^4$;
3) $93084 = 9^5 + 3^5 + 0^5 + 8^5 + 4^5$;

Write a Python program that;

1) Takes a positive integer number from the user;
2) Checks the entered number if it is Armstrong;
3) Consider the negative, float and any entries other than numeric values than display a warning message to the user.

∇ NOTE ⇒ Armstrong numbers ⇒ 407
153

SOLVE

```
n = ?nput ("Enter an ?nteger: ")
sum = 0

for ? ?n range (len(n)):
    sum += ?nt(n[?]) **len(n)

?f sum == ?nt(n):
    print(n, "an armstrong number")
elif float or ?nt or n<=0;
    print("It's an ?nvalid entry. Don't use
non numeric, float, or negative values.")

else:
    print("It's not an armstrong number")
```

TASK → Cevabı google'dan aldım. Ama sanki sadece 3 basamaklılar için geçerli.

(*) Armstrong number

```
num = int(input("Enter a number: "))

sum = 0

temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp //= 10

if num == sum:
    print(num, "is an Armstrong number")
elif (num == float) or (num == str) or (num <= 0):
    print("It's an invaled entry. Don't use numeric,
        float, or negative values.")
else:
    print(num, "is not an Armstrong number")
```

TASK 1

(*) PRIME NUMBER

Write a program that takes a number from the user and prints the result to check if it is a prime number.

```
num = int(input("Enter a number : "))
flag = false
if num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            flag = True
            break
if flag:
    print(num, "is not a prime number")
else:
    print(num, "is a prime number")
```

LOOPS (DÖNGÜLER)

1. for Loop
2. while Loop

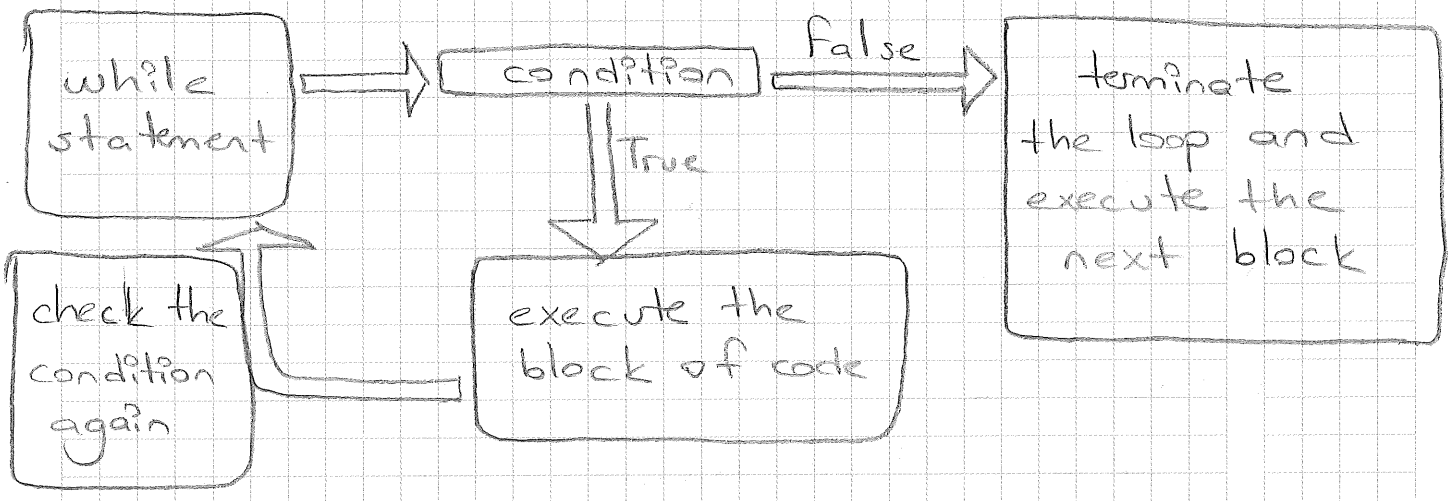
The loops are used to repeat (iterate) the execution of a block of code. Loops enable programmers to set certain sections of their code to repeat through a number of loops which are referred to as iterations.

WHILE Loop

while loops have a boolean logic, similar to if statements. As long as the result of the condition returns True, the code block under while loop runs.

When the condition returns to False, the loop execution is terminated and the program control moves further to the next operation. Here is the simple structure of a while loop:

```
while condition:
    body
```



We will not use this loop as often as the for loop, but it is still worth understanding.

(*)
```
number = 0
while number < 6 :
    print (number)
    number += 1
print ('now, number is bigger or equal to 6')
```

(*)
```
my_list = ["a", "b", "c", "d", "e"]
a=0
while a < len(my_list):
    print ('square of {} is: {}'.format(a, a**2))
    a+=1
```

'Guessing a number game'

```
answer = 44
question = 'what number am I thinking of?'
print("Let's play the guessing game!")

while True:
    guess = int(input(question))

    if guess < answer:
        print('Little higher')
    elif guess > answer:
        print('Little lower')
    else:    # guess == answer
        print('Are you a MINDREADER!!!')
        break
```

TASK

Fill in the blanks to complete the program that prints all elements of the flowers list below each on seperate lines such as:

Rose
Orchid
Tulip

```
flowers = ['Rose', 'Orchid', 'Tulip']
  count1 = len(...)
  count2 = ...
    while count... > 0:
      print(flowers[...])
        count... -= ...
        count... += ...
```

FOR Loop

When you want to iterate a block of code you will use for loop. To create a for loop, you need a variable and an iterable and an iterable object. Here is the simple structure of a for loop:

```
for variable in iterable:
    code block
```

(*)
```
for i in [1, 2, 3, 4, 5]:
    print(i)
```

→
```
1
2
3
4
5
```

* seasons = ['spring', 'summer', 'autumn', 'winter']

```
for season in seasons:
    print(season)
```

→ spring
summer
autumn
winter

* for i in {'n1': 'one', 'n2': 'two'} : print(i)

→ n1
n2

INTERVIEW Q&A

How does for loop and while loop differ in Python and when do you choose to use them?

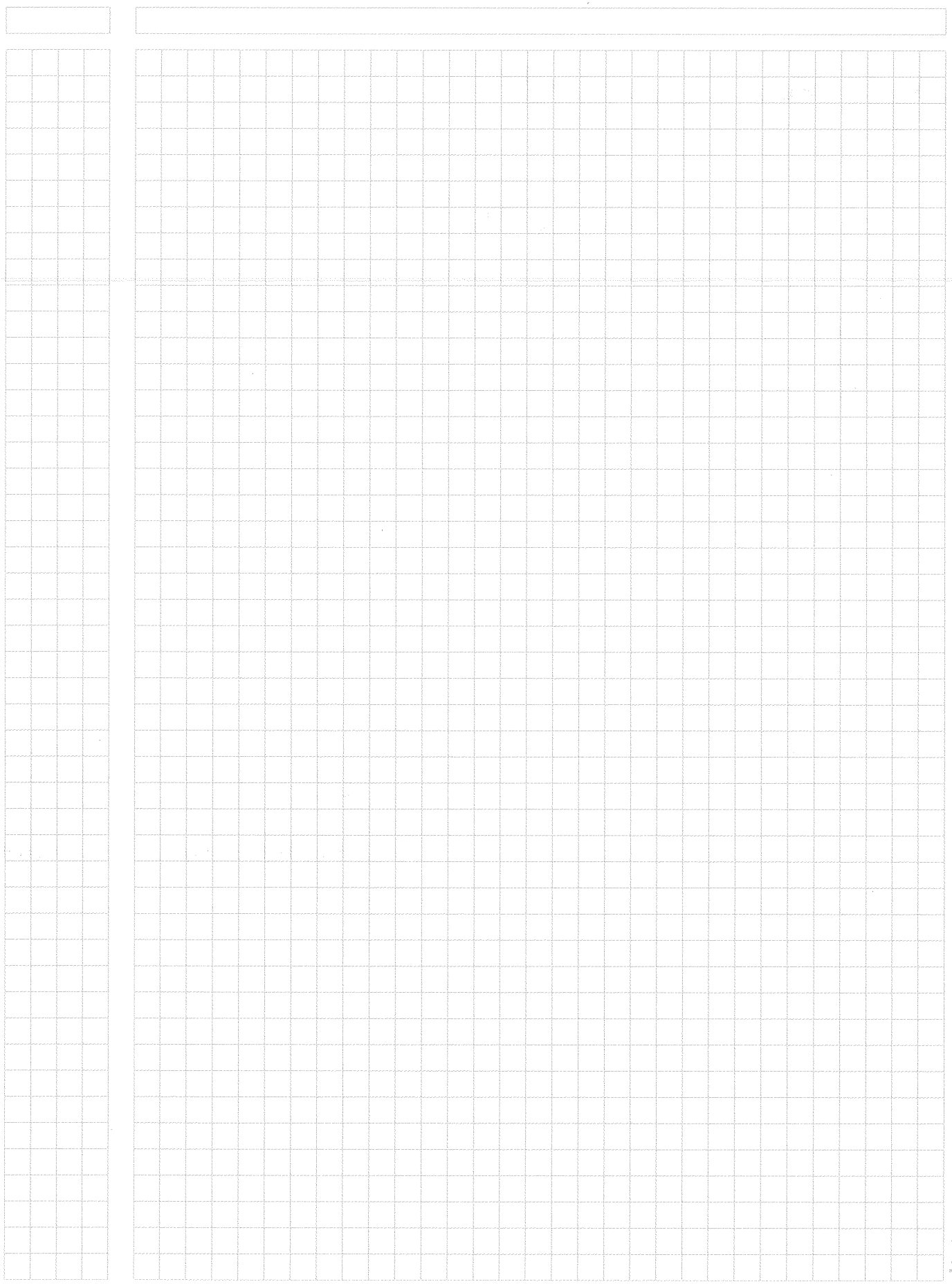
For loop is generally used to iterate through the elements of various collection types such as list, tuple, set and dictionary.

While loop is the actual looping feature that is used in any other programming language. This is how Python differs in handling loops from the other programming languages.

(*) iterable = [1, 2, 3, 4]

```
for i in iterable:
    print('{}'.format(i**2))
```

→
```
1
4
9
16
```



tonguç

tonguç

tonguç

tonguç

with Marcus