اصول مقدماتی پایتون 5

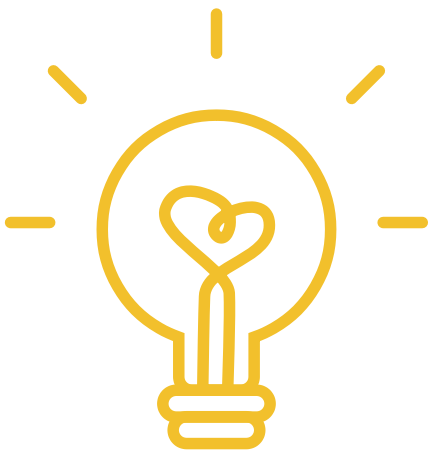
more types

هادی فرهادی

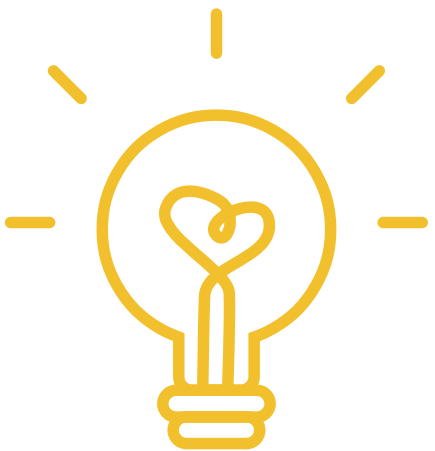
مهر ۱۴۰۴

h.farhadi.py@gmail.com

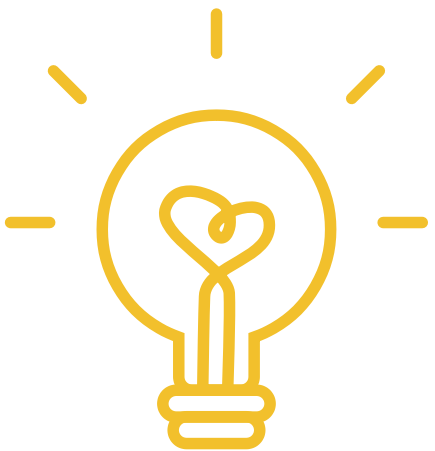
یادآوری



سؤال جواب



نوع str



نوع str

```
user_name: str = "hadi_main_py"
full_name: str = "Hadi Farhadi"

print("String Index".center(50, '-'))
# index - اندیس
print(user_name[2]) # output: d
print(user_name[-1]) # output: y

print("String Slicing".center(50, '-'))
# slicing - برش
print(user_name[0:6]) # output: hadi_m

print("String length".center(50, '-'))
# length - طول رشته
print(len(user_name))

# f-strings
print("String concat".center(50, '-'))
message: str = "Hello " + full_name + "!" # bad performance
message_with_f_string: str = f"Hello {full_name}!"
print(message)
print(message_with_f_string)
```

نوع str

```
user_name: str = "hadi_main_py"
full_name: str = "\tHadi Farhadi     "

print("String upper".center(50, '-'))
# upper - بزرگ کردن حروف
print(user_name.upper()) # output: HADI_MAIN_PY


print("String lower".center(50, '-'))
# lower - کوچک کردن حروف
print(user_name.lower()) # output: hadi_main_py

print("String strip".center(50, '-'))
# strip - حذف فاصله ها

print(full_name) # output:  Hadi Farhadi
print(full_name.strip()) # output: Hadi Farhadi

print("String startswith".center(50, '-'))
# startswith - شروع شدن با

print(user_name.startswith('hadi')) # output: True

print("String endswith".center(50, '-'))
# endswith - پایان با

print(user_name.endswith('txt')) # output: False

print("String in".center(50, '-'))
# in

print("py" in user_name) # output: True
```

نوع str

اتصال رشته ها توسط اپراتور + و join

```
print("String concatenation with +".center(50, "-"))
str_list: list = ["Hello", "World", "What is this?"]

str_concat_with_plus: str = ""

for item in str_list:
    str_concat_with_plus += item + " "

print(str_concat_with_plus)


print("String concatenation with join".center(50, "-"))
str_concat_with_join: str = " ".join(str_list)
print(str_concat_with_join)
```

نوع str

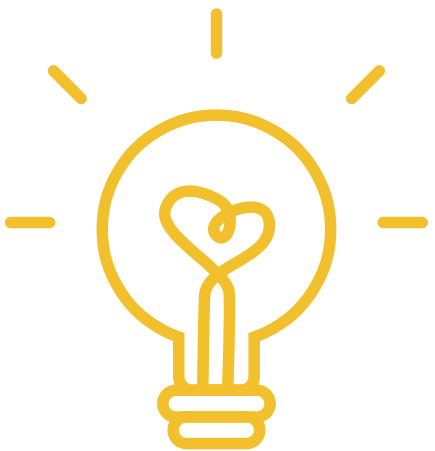
```
# format

print("String format".center(50,"-"))
login_code_message: str = "Hello {0}! How are you? Your login code is {code}"

print(login_code_message) # output: Hello {0}! How are you? Your login code is {code}

print(login_code_message.format("Ali", code="Hello")) # output: Hello Ali! How are you? Your login
code is Hello
```

نوع str

تعداد space های یک رشته

```
full_name: str = "\tHadi Farhadi     "

print("Space count in full_name".center(50, '-'))

# تعداد فاصله های داخل نام

space_count: int = 0
for letter in full_name:
    if letter == " ":
        space_count += 1

print(f"{space_count} spaces in {full_name}")
```

نوع tuple



نوع tuple

تاپل یک نوع داده‌ی sequence در پایتون است که شبیه لیست است،

اما با چند تفاوت مهم. تاپل immutable (تغییرناپذیر) است و با پرانتز () تعریف می‌شود.

بعد از انتساب اولیه نمی توان مقدار آن را تغییر داد.

سریعتر از لیست است و حافظه ی کمتری مصرف می کند.

```
empty_tuple: tuple = ()
print(empty_tuple)


single_item = (42,)  # () is required
print(single_item)   # output: (42,)


fruits = ("apple", "banana", "cherry")
numbers = (1, 2, 3, 4, 5)
mixed = (1, "hello", 3.14, True, [5, 3], (22, 11))
```

نوع tuple

```
fruits = ("apple", "banana", "cherry")
print(fruits)

# index
print(fruits[0])
print(fruits[-1])

# Error: changing its value
fruits[0] = "kiwi"
print(fruits)


numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
print(numbers[2:5])   # output: (2, 3, 4)
print(numbers[:3])    # output: (0, 1, 2)
print(numbers[7:])    # output: (7, 8, 9)
print(numbers[::3])   # output: (0, 3, 6, 9)

print(numbers.count(3))

print("apple" in fruits)  # output: True
print("orange" in fruits) # output: False

print(len(fruits))  # output: 3

print(fruits.index("banana")) # output: 1
```

نوع tuple

```
# Constant
USERS = ((1, "Nima Rabbani", "Male", 32), (2, "Fatemeh Rajabi", "Female", 28),
         (3, "Hanieh Bahrami", "Female", 35), (4, "Rashed Ragheb", "Male", 42))


input_user = [3, "Hanieh Bahrami", "Female", 35]

if input_user in USERS:
    print(f"{input_user} is in the USERS list")
else:
    print(f"{input_user} is not in the USERS list")


input_user_tuple = tuple(input_user)

if input_user_tuple in USERS:
    print(f"{input_user} is in the USERS list")
else:
    print(f"{input_user} is not in the USERS list")


if input_user_tuple == USERS[2]:
    print(f"{input_user} is in the USERS list")
```

نوع tuple

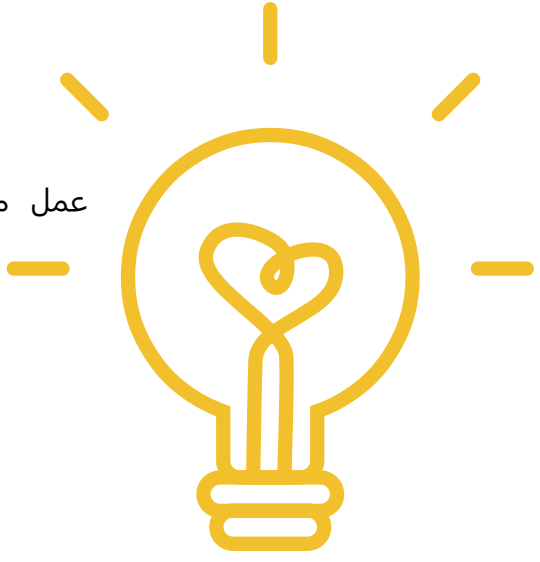
```
# مرتب سازی یک tuple
USERS = ((1, "Nima Rabbani", "Male", 32), (2, "Fatemeh Rajabi", "Female", 28),
         (3, "Hanieh Bahrami", "Female", 35), (4, "Rashed Ragheb", "Male", 42))


def sort_users(users, sort_by = 0, reverse = False):
    sorted_users = sorted(users, key=lambda user: user[sort_by], reverse=reverse)
    return sorted_users

sorted_users = sort_users(USERS, sort_by=1, reverse=True)

print("Sorted USERS(tuple)".center(50, '-'))
print("id\tfull_name\tsex\tage")
for user in sorted_users:
    print(f"{user[0]}\t{user[1]}\t{user[2]}\t{user[3]}")
```

نوع tuple

```
# مقایسه tuple

first_tuple = (1, "Hello", True)
second_tuple = (2, "Hello", False)

if first_tuple >= second_tuple:
    print("First tuple >= second tuple")

first_tuple = (1, "Hello", True)
second_tuple = (1, "Hallo", True)

if first_tuple >= second_tuple:
    print("First tuple >= second tuple")

first_tuple = (1, 3.25, "LTS")
second_tuple = (1, "Hello", False)
if first_tuple >= second_tuple: # raise an exception
    print("First tuple >= second tuple")
```

عمل مقایسه فقط بین عناصر از یک نوع انجام می شود

پس در دنباله ها عملیات مقایسه تا زمانی که بتوان بین عناصر متناظر، مقایسه ای انجام داد ادامه می یابد

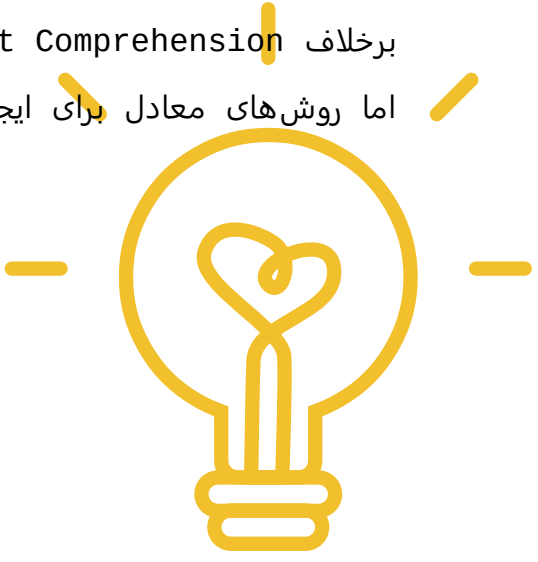


نوع tuple

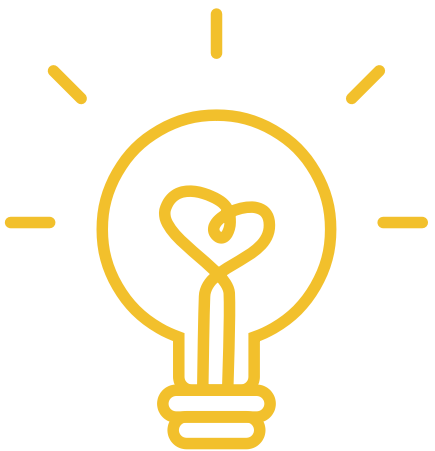
```
# tuple comprehension


numbers: list[int] = [1, 2, 3, 4, 5, 6, 7, 8, 9]

result: tuple[int] = tuple(number**2 for number in numbers if number % 2 != 0) # tuple is required

print("Tuple comprehension".center(50, "-"))
print(result)  # {1, 9, 25, 49, 81}
```

برخلاف List Comprehension و Set Comprehension، پایتون Tuple Comprehension به صورت مستقیم ندارد.

اما روش‌های معادل برای ایجاد tuple با استفاده از generator expressions وجود دارد.



نوع dict



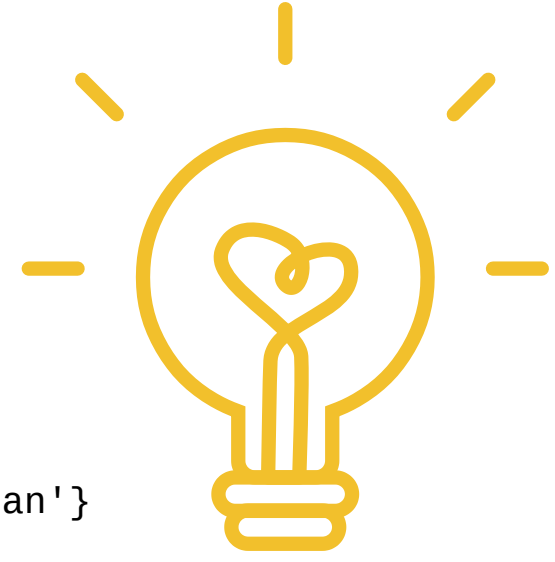
نوع dict

دیکشنری یک ساختار داده‌ای در پایتون است که داده‌ها را

به صورت جفت‌های کلید-مقدار (key-value) ذخیره می‌کند.

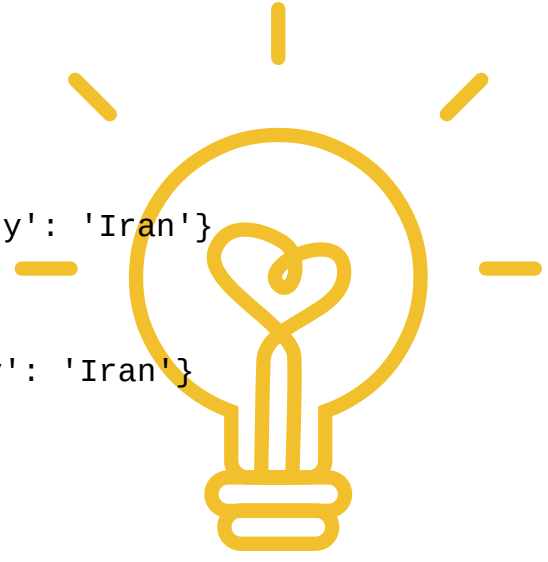
دیکشنری با استفاده از آکولاد {} تعریف می‌شود و هر کلید با یک مقدار مرتبط است.

```
empty_dict: dict = {}
print(empty_dict)  # output: {}


person: dict = {
    "name": "Ali",
    "age": 30,
    "city": "Tehran"
}

print(person)  # output: {'name': 'Ali', 'age': 30, 'city': 'Tehran'}
```



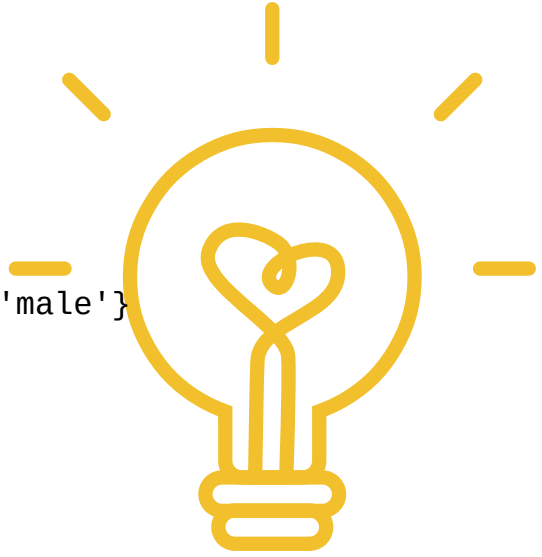
نوع dict

```
person: dict[str, any] = {
    "name": "Ali",
    "age": 30,
    "city": "Tehran"
}

print(person)  # output: {'name': 'Ali', 'age': 30, 'city': 'Tehran'}

# index
print(person["name"]) # output: Ali
print(person.get("name"))  # output: Ali
print(person.get("country", "Iran"))

# change its value
person["age"] = 31
print(person)  # output: {'name': 'Ali', 'age': 31, 'city': 'Tehran'}

# add new item
person["country"] = "Iran"
print(person)  # output: {'name': 'Ali', 'age': 31, 'city': 'Tehran', 'country': 'Iran'}

# update value
person["city"] = "London"
print(person) # output: {'name': 'Ali', 'age': 31, 'city': 'London', 'country': 'Iran'}

# remove an item
del person["city"]
print(person)  # output: {'name': 'Ali', 'age': 31, 'country': 'Iran'}
```



نوع dict

```
# pop an item
age = person.pop("age")
print(age)     # output: 31
print(person)  # output: {'name': 'Ali', 'country': 'Iran'}

# keys
print(person.keys())  # output: dict_keys(['name', 'country'])

# values
print(person.values()) # output: dict_values(['Ali', 'Iran'])

# items
print(person.items())  # output: dict_items([('name', 'Ali'), ('country', 'Iran')])

# copy without reference
person_copy = person.copy()
print(person_copy)  # output: {'name': 'Ali', 'country': 'Iran'}


# update multiple items
person.update({"age": 38, "country": "USA", "sex": "male"})
print(person)  # output: {'name': 'Ali', 'age': 38, 'country': 'USA', 'sex': 'male'}

# clear dict
person.clear()
print(person)  # output: {}
```



نوع dict

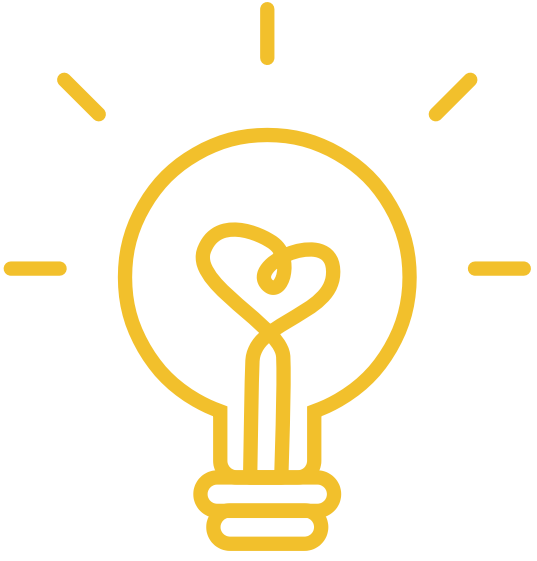
Nested dict دیکشنری تو در تو

```
user: dict = {
    "username": "ali_azimi",
    "email": "ali_azimi@example.com",
    "password": "123456",
    "profile": {
        "first_name": "Ali",
        "last_name": "Azimi",
        "age": 33,
        "sex": "male",
    }
}

print("User nested dictionary".center(50, "-"))
print(user["username"])
print(user["profile"]["first_name"])
print(user["profile"]["last_name"])
print(user["profile"]["age"])
print(user["profile"]["sex"])
```

نوع dict

محاسبه تعداد تکرار کاراکترها در یک رشته ورودی

```
print("Counting letters in a string".center(50, "-"))

input_str: str = input("Enter a string: ")

letters_count = {}

for letter in input_str:
    letters_count[letter] = letters_count.get(letter, 0) + 1

for letter, count in letters_count.items():
    print(f"{letter}: {count}")
```



نوع dict

تبدیل تاپل به لیست از دیکشنری

```
# id, full_name, sex, age
USERS = ((1, "Nima Rabbani", "Male", 32), (2, "Fatemeh Rajabi", "Female", 28),
         (3, "Hanieh Bahrami", "Female", 35), (4, "Rashed Ragheb", "Male", 42))

user_list = []
for user in USERS:
    user_dict: dict = {
        "id": user[0],
        "full_name": user[1],
        "sex": user[2],
        "age": user[3]
    }
    user_list.append(user_dict)

print("Users".center(50, "-"))

print("id\tfull_name\tsex\tage")
for user in user_list:
    print(user["id"], user["full_name"], user["sex"], user["age"], sep="\t")
```

نوع dict

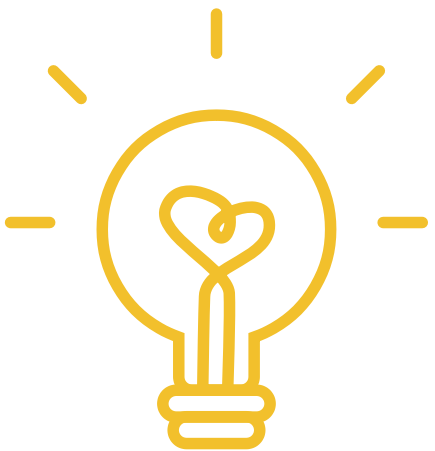
تبدیل تاپل به لیستی از دیکشنری

```
# id, full_name, sex, age
USERS = ((1, "Nima Rabbani", "Male", 32), (2, "Fatemeh Rajabi", "Female", 28),
         (3, "Hanieh Bahrami", "Female", 35), (4, "Rashed Ragheb", "Male", 42))

# list comperihension
user_list = [{"id": user[0], "full_name": user[1], "sex": user[2], "age": user[3]} for user in USERS]

print("Users".center(50, "-"))

print("id\tfull_name\tsex\tage")
for user in user_list:
    print(user["id"], user["full_name"], user["sex"], user["age"], sep="\t")
```

نوع dict

```
# dictionary comperhension


number_list = [1,2,3,4,5,6,7,8,9]

user_info_dict = { number: number ** 2 for number in number_list }

print("Dictonary comprehension".center(50, '-'))
for key, value in user_info_dict.items():
    print(f"{key} = {value}")
```

نوع list



نوع list

list comperhension

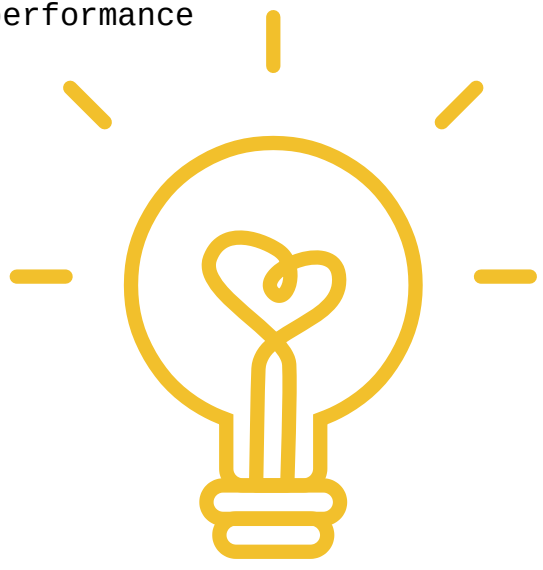
```
# Gold forex data (Open, Close, Low, High)
gold_data = [
    [1954.32, 1956.78, 1953.15, 1957.24],
    [1956.78, 1958.45, 1955.62, 1959.83],
    [1958.45, 1957.23, 1956.11, 1959.02],
    [1957.23, 1955.67, 1954.89, 1958.15],
    [1955.67, 1953.42, 1952.18, 1956.34],
    [1953.42, 1956.89, 1952.75, 1957.45],
    [1956.89, 1959.34, 1955.92, 1960.27]
]

print(gold_data[1][3]) # output

low_list = [row[2] for row in gold_data]
high_list = [row[3] for row in gold_data]

print("low price".center(50, "-"))
for low in low_list:
    print(f"{low:.1f}")

print("high price".center(50, "-"))
for high in high_list:
    print(f"{high:.3f}")
```

نوع list

```
# nested for on 2d list


# Gold forex data (Open, Close, Low, High)
gold_data = [
    [1954.32, 1956.78, 1953.15, 1957.24],
    [1956.78, 1958.45, 1955.62, 1959.83],
    [1958.45, 1957.23, 1956.11, 1959.02],
    [1957.23, 1955.67, 1954.89, 1958.15],
    [1955.67, 1953.42, 1952.18, 1956.34],
    [1953.42, 1956.89, 1952.75, 1957.45],
    [1956.89, 1959.34, 1955.92, 1960.27],
    [1959.34, 1961.78, 1958.46, 1962.53]
]

print("nested for".center(50, "-"))

for row in gold_data:
    for col in row:
        print(f"{col:.2f}", end=" ")
    print()
```

الحاق آیتم های دو لیست رشته ای با while

```
first_name_list = ["Hadi", "Mahdi", "Hamid", "Sara", "Shahla", "Roya"]
last_name_list = ["Farhadi", "Rajabi", "Sadri", "Sorian", "Ahmadi", "Dehghan"]

count = 0

# size = min(len(first_name_list), len(last_name_list)) # the best approach

user_full_name_list = []
while count < len(first_name_list):
    # full_name = first_name_list[count] + " " + last_name_list[count] # bad performance
    # full_name = f"{first_name_list[count]} {last_name_list[count]}"
    full_name = " ".join([first_name_list[count], last_name_list[count]])
    user_full_name_list.append(full_name)

    count += 1

print("join two list with while".center(50, "-"))

for name in user_full_name_list:
    print(name)
```



نوع list

الحاق آیتم های دو لیست رشته ای با for

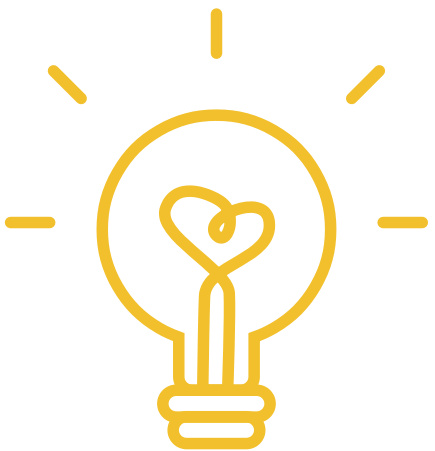
```
first_name_list = ["Hadi", "Mahdi", "Hamid", "Sara", "Shahla", "Roya"]
last_name_list = ["Farhadi", "Rajabi", "Sadri", "Sorian", "Ahmadi", "Dehghan"]

# size = min(len(first_name_list), len(last_name_list)) # the best approach

user_full_name_list = []
for index in range(len(first_name_list)):
    # full_name = first_name_list[count] + " " + last_name_list[count] # bad performance
    # full_name = f"{first_name_list[count]} {last_name_list[count]}"
    full_name = " ".join([first_name_list[index], last_name_list[index]])
    user_full_name_list.append(full_name)

print("join two list with for".center(50, "-"))

for name in user_full_name_list:
    print(name)
```

نوع list

الحاق آیتم های دو لیست رشته ای با zip

```
first_name_list = ["Hadi", "Mahdi", "Hamid", "Sara", "Shahla", "Roya"]
last_name_list = ["Farhadi", "Rajabi", "Sadri", "Sorian", "Ahmadi", "Dehghan"]

user_full_name_list = [f"{first_name} {last_name}" for first_name,
                           last_name in zip(first_name_list, last_name_list)]

print("join two list with zip".center(50, "-"))

for name in user_full_name_list:
    print(name)
```

نوع set



نوع set

Set یک نوع داده در پایتون است که مجموعه‌ای از

عناصر منحصربفرد و بدون ترتیب را ذخیره می‌کند.

عناصر تکراری در set مجاز نمی باشد

قابل تغییر (mutable) است

```
empty_set: set = set()
print(empty_set)

number_set = {1, 2, 3, 4, 5, 5, 7, 8, 8, 10}
print(number_set) # output: {1, 2, 3, 4, 5, 7, 8, 10}
```

نوع set

```
# حذف داده های تکراری از لیست

number_list: list = [1, 3, 3, 8, 9, 4, 5, 7, 4, 3]

print("list with duplicate items".center(50, '-'))
print(number_list)

number_list_without_duplicate: list = []
for number in number_list:
    if number not in number_list_without_duplicate:
        number_list_without_duplicate.append(number)

print("list without duplicate items".center(50, '-'))
print(number_list_without_duplicate)
```

نوع set

```
# حذف داده های تکراری از لیست با set

number_list: list = [1, 3, 3, 8, 9, 4, 5, 7, 4, 3]

print("list with duplicate items".center(50, '-'))
print(number_list)

number_set: set = set(number_list)
number_list_without_duplicate: list = list(number_set)

print("list without duplicate items".center(50, '-'))
print(number_list_without_duplicate)
```

نوع set

مهم بودن ترتیب در لیست ها

```
first_number_list: list = [1, 2, 3]
second_number_list: list = [3, 2, 1]
third_number_list: list = [1, 2, 3]

print(first_number_list == second_number_list) # output: False
print(first_number_list == third_number_list) # output: True
```

نوع set

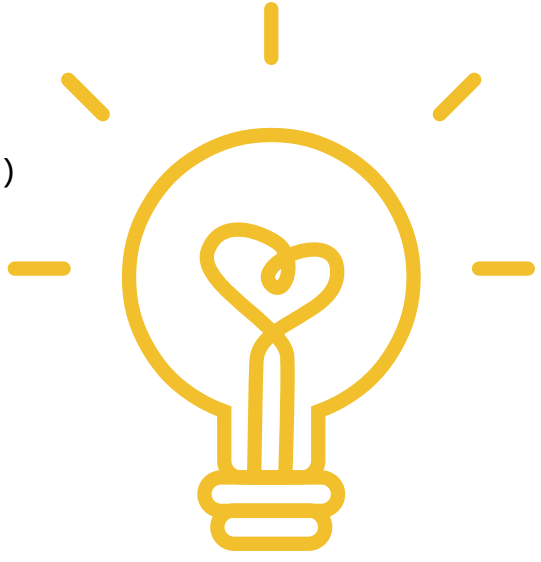
مهم نبودن ترتیب در لیست ها

```
first_number_set: set = {1, 2, 3}
second_number_set: set = {3, 2, 1}

third_number_set: set = {1, 2, 3, 2, 3, 1, 2}


print(first_number_set == second_number_set) # output: True

print(first_number_set == third_number_set) # output: ?
```

نوع set

اجتماع - union

```
first_number_set: set = {1, 2, 3}
second_number_set: set = {3, 4, 5}

print(f"first_number_set: {first_number_set}")
print(f"second_number_set: {second_number_set}")

# اجتماع - union

print("Union".center(50, "-"))

union_set = first_number_set | second_number_set
print(f"union with | operator: {union_set}") # output: {1, 2, 3, 4, 5}

print(f"union with union method: {first_number_set.union(second_number_set)}")
```



نوع set

اشتراک - intersection

```
first_number_set: set = {1, 2, 3}
second_number_set: set = {3, 4, 5}

print(f"first_number_set: {first_number_set}")
print(f"second_number_set: {second_number_set}")

# اشتراک - intersection

print("Intersection".center(50, "-"))

intersection_set = first_number_set & second_number_set
print(f"intersection with & operator: {intersection_set}")

print(f"intersection with intersection method:
{first_number_set.intersection(second_number_set)}")
```

نوع set

تفریق - difference

```
first_number_set: set = {1, 2, 3}
second_number_set: set = {3, 4, 5}

print(f"first_number_set: {first_number_set}")
print(f"second_number_set: {second_number_set}")

# تفریق - difference

print("Difference".center(50, "-"))

difference_set = first_number_set - second_number_set
print(f"difference with - operator: {difference_set}")

print(f"difference with difference method:
{first_number_set.difference(second_number_set)}")
```

نوع set

```
name_set: set = set()

name_set.add("John")
name_set.add("Erfan")
name_set.add("Leila")
name_set.add("Ali")
name_set.add("Vahid")
name_set.add("Karin")


print("Names".center(50, "-"))
for (index, name) in enumerate(name_set, start=1):
    print(f"{index} {name}")

print("in operator".center(50, "-"))
if "Hadi" in name_set:
    print(f"Where are you now, Hadi?")

student_set: set = {"Erfan", "Leila", "Vahid"}

print("Subset".center(50, "-"))
print(f"{student_set.issubset(name_set)}")

print("Remove".center(50, "-"))
name_set.remove("Hadi")
print(name_set)

print("Index".center(50, "-"))
print(name_set[0]) # Error
```

نوع set

set comprehension

```
numbers: list[int] = [1, 2, 3, 4, 5, 6, 7, 8, 9]
result: set[int] = {number**2 for number in numbers if x % 2 != 0}

print("Set comprehension".center(50, "-"))
print(result)  # output: {1, 9, 25, 49, 81}
```

built in functions for iterators

built in functions for iterators

all: اگر همه آیتم ها مقدار داشته باشند True را بر می گرداند در غیر اینصورت False را بر می گرداند

خروجی این تابع همیشه bool است یعنی True یا False است

```
# all function

print("all function".center(50, "-"))

number_list: list = [0, 1, 2, 3, 4]
print(all(number_list))

message_seen_list: list = [True, False, False, True]
print(all(message_seen_list))
```

built in functions for iterators

any: اگر حداقل یکی از آیتم ها مقدار داشته باشند True را بر می گرداند در غیر اینصورت False را بر می گرداند

خروجی این تابع همیشه bool است یعنی True یا False است

```
# any function

print("any function".center(50, "-"))

number_list: list = [0, 1, 2, 3, 4]
print(any(number_list)) # output: True

message_seen_list: list = [True, False, False, True]
print(any(message_seen_list)) # output: True
```

built in functions for iterators

ترکیب چند تابع با هم

```
point_list = [5, 2, 8, 1, 9, 3]

# مرتب کردن، معکوس کردن و گرفتن 3 عدد بزرگتر
result = list(reversed(sorted(point_list)))[:3]
print(result)  # [9, 8, 5]
```

Call by value, Call by reference

Immutable Objects (اشیاء تغییرناپذیر)

پس از ایجاد نمی‌توانند تغییر کنند

اگر سعی در تغییر آنها داشته باشید، یک شیء جدید ایجاد می‌شود

مثال‌ها:

اعداد (int, float)

رشته‌ها (str)

تاپل‌ها (tuple)

بایت‌های تغییرناپذیر (bytes)

Boolean (bool)

Mutable Objects (اشیاء تغییرپذیر)

Mutable Objects (اشیاء تغییرپذیر)
پس از ایجاد می‌توانند تغییر کنند

تغییرات روی همان شیء اصلی اعمال می‌شود

مثال‌ها:

لیست‌ها (list)

دیکشنری‌ها (dict)

مجموعه‌ها (set)

بایت‌های تغییرپذیر (bytearray)

Call by Object Reference

پایتون از مدل "Call by Object Reference" استفاده می‌کند که ترکیبی از Call by Value و Call by Reference است:

برای Immutable Objects: مانند Call by Value عمل می‌کند

برای Mutable Objects: مانند Call by Reference عمل می‌کند

```
def process_data(mutable_list: list, immutable_param: int):
    mutable_list.append([66, 77, 99])

    immutable_param += 100


two_d_list: list = [[1, 2, 3], [3, 4, 5], [7, 8, 9]]
number: int = 10

process_data(two_d_list, number)
print(two_d_list)  # the value has changed
print(number)  # it hasn't changed
```

Value type and Reference type

```python
first_str: str = 'Hello World!'
second_str: str = 'Hello World!'

print(first_str is second_str)
print(id(first_str), id(second_str))

first_number_list: list = [1, 2, 3]
second_number_list: list = [1, 2, 3]
third_number_list: list = first_number_list

print(first_number_list is second_number_list)
print(id(first_number_list), id(second_number_list))
print(third_number_list is first_number_list)
print(id(first_number_list), id(third_number_list))


first_number_list.append(4)
print(first_number_list) # [1, 2, 3, 4]
print(third_number_list) # [1, 2, 3, 4]

third_str: str = first_str
first_str += "88"

print(first_str) # Hello World!88
print(third_str) # Hello World!
```

Value type and Reference type

```
number: int = 32

print(number)
print(id(number))

number += 32

print(number)
print(id(number))

number_list: list = [1, 2, 3]
print(number_list)
print(id(number_list))

number_list += [4, 5, 6]
print(number_list)
print(id(number_list))
```

hash table

hash table

مفهوم پایه‌ای Hash Table

Hash Table یک ساختار داده‌ای است که داده‌ها را به صورت جفت‌های کلید-مقدار (Key-Value) ذخیره می‌کند.

این ساختار برای دسترسی سریع به داده‌ها طراحی شده است (میانگین زمان دسترسی برای درج و حذف در شرایط ایده آل: O(1) و در بدترین حالت O(n)).

Collision زمانی رخ می‌دهد که دو کلید مختلف مقدار hash یکسان تولید کنند. این بدترین حالت است در غیر اینصورت شرایط ایده آل است.

در پایتون، Hash Table به صورت دیکشنری (Dictionary) و Set پیاده‌سازی شده است.

هر کلید به یک عدد منحصربه‌فرد (hash value) تبدیل می‌شود.

خروجی: یک عدد صحیح مانند 123456789 # hash("hello")

Set در پایتون یک Hash Table است که فقط کلید ذخیره می‌کند (بدون مقدار).

در نسخه‌های قدیمی پایتون (<3.7)، دیکشنری ترتیب را حفظ نمی‌کرد. اما از پایتون 3.7 به بعد، ترتیب درج حفظ می‌شود.

copy

copy

در پایتون، هنگام کار با اشیاء mutable مانند لیست‌ها و دیکشنری‌ها، درک تفاوت بین shallow copy و deep copy بسیار مهم است. این مفهوم به چگونگی کپی کردن اشیاء و روابط بین آن‌ها مربوط می‌شود.

Shallow Copy (کپی سطحی)

یک shallow copy شیء جدید ایجاد می‌کند، اما به جای کپی کردن اشیاء تو در تو، فقط reference آن‌ها را کپی می‌کند.

Shallow Copy (کپی سطحی)

```
import copy

numbers: list = [1, 2, 3, [4, 5, 6]]
print("numbers".center(30, "-"))
print(numbers)

numbers_copy: list = numbers
print("numbers_copy".center(30, "-"))
print(numbers_copy)

print("Copy a list with assignment".center(50, '-'))
numbers_copy.append((1, True))

print(numbers)
print(numbers_copy)

print("Copy a list with copy method".center(50, '-'))
numbers_copy_with_copy_method: list = numbers.copy()
# numbers_copy_with_slicing: list = numbers[:]
# numbers_copy_with_copy_module: list = copy.copy(numbers)

numbers_copy_with_copy_method.append("Thank you")

print(numbers)
print(numbers_copy_with_copy_method)

print("Change nested list - copy method".center(50, '-'))
numbers_copy_with_copy_method[3].append(3.5)

print(numbers)
print(numbers_copy_with_copy_method)
```

تغییرات در اشیاء سطح اول مستقل هستند

تغییرات در اشیاء تو در تو بر هر دو نسخه تاثیر می‌گذارند

Deep Copy (کپی عمیق)

یک deep copy شیء جدید ایجاد می‌کند و به صورت **recursive** تمام اشیاء تو در تو را نیز کپی می‌کند.

Deep Copy (کپی عمیق)

```
import copy

numbers: list = [1, 2, 3, [4, 5, 6]]
print("numbers".center(30, "-"))
print(numbers)

numbers_copy: list = numbers
print("numbers_copy".center(30, "-"))
print(numbers_copy)

print("Copy a list with deepcopy method".center(50, '-'))
numbers_copy_with_copy_module: list = copy.deepcopy(numbers)

numbers_copy_with_copy_module.append("Thank you")
numbers_copy_with_copy_module[3].append(3.5)

print(numbers)
print(numbers_copy_with_copy_module)
```

تغییرات در تمام سطوح مستقل هستند

هیچ ارتباطی بین نسخه اصلی و کپی شده وجود ندارد

هزینه عملکرد

Shallow Copy: سریع‌تر و کم‌مصرف‌تر از نظر حافظه

Deep Copy: کندتر و پرمصرف‌تر از نظر حافظه (به خصوص برای ساختارهای بزرگ)

این مثال ها برای لیست بودند. برای دیکشنری هم به همین صورت است. برای برد گیم ها خیلی کاربرد دارند

Mini Project

Library Book

```
# id, title, author, publish date, gener, available

BOOKS = (
    (1, "داستان" ,1943 ,"آنتوان دو سنت اگزوپری" ,"شازده کوچولو", True),
    (2, "1984", "علمی-تخیلی" ,1949 ,"جورج اورول", True),
    (3, "اقتصاد" ,1776 ,"آدام اسمیت" ,"ثروت ملل", False),
    (4, "تاریخ" ,2011 ,"یووال نوح هراری" ,"انسان خردمند", True),
    (5, "داستان" ,1988 ,"پائولو کوئلیو" ,"کیمیاگر", True),
    (6, "داستان" ,1967 ,"گابریل گارسیا مارکز" ,"صد سال تنهایی", False),
    (7, "فانتزی" ,1997 ,"جی. کی. رولینگ" ,"هری پاتر و سنگ جادو", True),
    (8, "تاریخ" ,1925 ,"آدولف هیتلر" ,"نبرد من", True),
    (9, "اقتصاد" ,1997 ,"رابرت کیوساکی" ,"پدر پولدار پدر بی‌پول", True),
    (10, "شعر" ,1390 ,"حافظ شیرازی" ,"دیوان حافظ", True)
)
```

وظایف:

- تبدیل کتاب ها به لیستی از دیکشنری توسط تابع
- مرتب سازی کتاب ها براساس تاریخ انتشار و آیدی
- نمایش کتاب های موجود
- اضافه کردن کتاب به لیست قرض داده شده و تغییر مقدار در دسترس بودن
- پاک کردن لیست کتاب هایی که قرض گرفته اید
- نمایش سبد کتاب هایی که قرض گرفته شده اند

THANK YOU

h.farhadi.py@gmail.com