تمرین و رفع اشکال

هادی فرهادی
مهر ۱۴۰۴

h.farhadi.py@gmail.com

Variable - متغیر

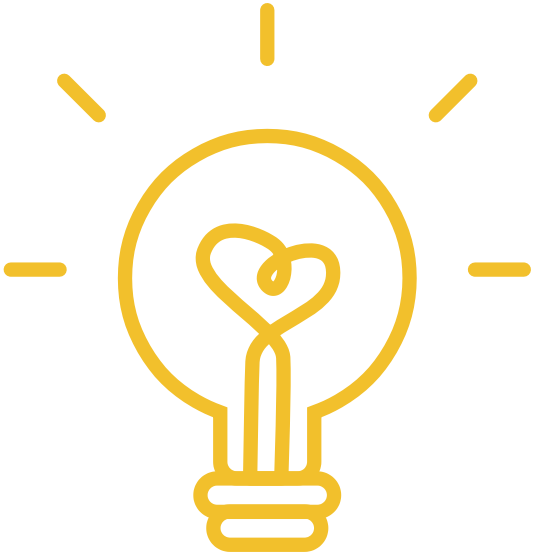
h.farhadi.py@gmail.com

Variable - متغیر

ظرفی است برای نگه داری چیزها.

علم کامپیوتر: مکانی در فضای ذخیره سازی (هارد، رم) است برای ذخیره داده‌ها،
که با یک نام قابل دسترسی است.

```
apple_basket = 5

first_name: str = “Ali”

full_name: str = “Reza Chamanzar”

points: float = 18.75

is_professor: bool = False
```



h.farhadi.py@gmail.com

تمرین: جابجایی مقادیر دو متغییر

h.farhadi.py@gmail.com

شرط‌ها در پایتون: if، else و elif

h.farhadi.py@gmail.com

شرط‌ها در پایتون: if، else و elif

در پایتون، از ساختارهای شرطی برای کنترل جریان
اجرای برنامه بر اساس شرایط مختلف استفاده می‌شود.
مثلاً در یوتوب اگر سن کاربر بیشتر از ۱۸ سالش بود این محتوا را می تونه
ببینه در غیر اینصورت نمی تونه ببینه

```
if condition1:
    # do this operation # 1
elif condition2:
    # do this operation # 2
elif condition3:
    # do this operation # 3
else:
    # do this operation # 4
```

h.farhadi.py@gmail.com

تو رفتگی بعد از : الزامی است

برای خوانایی بیشتر است اما اگه رعایت نشه به خطا می‌خورید

همیشه یکی از شرط ها درست هستند و اجرا می شوند

از تو رفتگی تا انتهای آن تو رفتگی = یک بلاک کد

h.farhadi.py@gmail.com

شرط‌های تو در تو (Nested If) در پایتون

h.farhadi.py@gmail.com

شرط‌های تو در تو (Nested If) در پایتون

شرط‌های تو در تو به معنای استفاده از دستورات شرطی در داخل یکدیگر است.این روش زمانی استفاده می‌شود که نیاز به بررسی شرایط چندلایه داریم.

```
if condition1:
    # condition1 == True, now we want to access some data and check that data
        if condition2:
                …
        elif condition3:
                …
elif condition4:
        ...
```

سعی کنید از Nested If فقط در صورت نیاز استفاده کنید

چون خوانایی کد را پایین می آورد

h.farhadi.py@gmail.com

استفاده از چند **condition** به جای **Nested if**

h.farhadi.py@gmail.com

استفاده از چند condition به جای Nested if

جایگزین‌ها: گاهی می‌توان با استفاده از عملگرهای منطقی (and, or) از شرط‌های تو در تو اجتناب کرد:

```
if condition1:
    if condition2:
        if condition3:
            ...

if condition1 and condition2 and condition3:
    ...
```

h.farhadi.py@gmail.com

در پایتون (List) لیست

h.farhadi.py@gmail.com

لیست (List) در پایتون

لیست‌ها (type) یکی از پرکاربردترین ساختارها
که برای ذخیره مجموعه‌ای از مقادیر استفاده می‌شوند.

تغییرپذیر (Mutable): می‌توان محتوای لیست را تغییر داد

```
empty_list: list = []
empty_list: list = list()

numbers: list = [1, 2, 3, 4, 5]
fruits: list = ["apple", "banana", "orange"]
mixed: list = [1, "hello", 3.14, True]
```

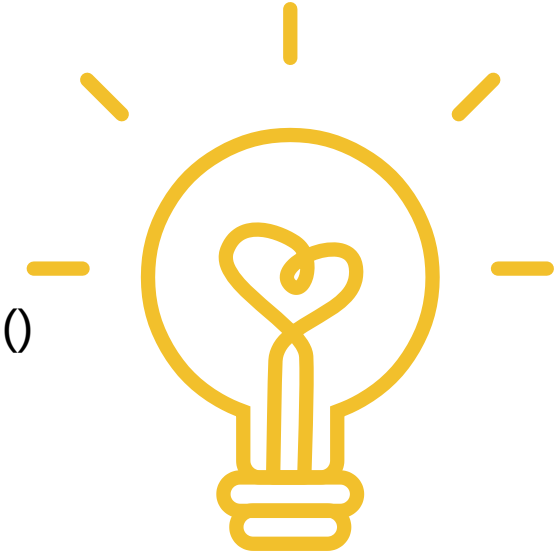
h.farhadi.py@gmail.com

اضافه و حذف کردن عناصر لیست

یکسری عملیات هست که از طریق کدهایی (متدها یا فانکشن هایی)
که برای آن نوع طراحی شده انجام می‌پذیرد.
به عنوان مثال:
تابع یا فانکشن print() که توسط خود پایتون طراحی شده است
و به ما در نمایش خروجی کمک می‌کند.
انواع یا Type ها (ذکر شده در درس قبل) هم شامل یکسری تابع یا فانکشن هستند
(که به آن‌ها متد گفته می‌شود) که توسط خود پایتون طراحی شده است
که از طریق نام متغییر و کاراکتر . قابل دستیابی هستند

```
fruits: list = ["apple", "banana"]
fruits.append("orange")
fruits.insert(1, "kiwi")
fruits.extend(["grape", "mango"])
fruits.remove("kiwi")
fruits.index(“banana”)
```

```
del fruits[1]  # حذف با اندیس
removed_item: str = fruits.pop()
fruits.clear()
fruits.count(“apple”)
```



h.farhadi.py@gmail.com

built in functions (list)

h.farhadi.py@gmail.com

built in functions (list)

```
numbers: list = [3, 1, 4, 1, 5, 9, 2, 6]

print(len(numbers))

print(min(numbers))

print(max(numbers))

print(sum(numbers))

print(sorted(numbers))
```

توابع max ، min() ، sorted() و sum()

لیست اصلی را تغییر نمی‌دهند

کارایی: این توابع بهینه‌سازی شده‌اند و برای لیست‌های بزرگ مناسب هستند

h.farhadi.py@gmail.com

تمرین: ساخت ساختار داده Stack(LIFO)

h.farhadi.py@gmail.com

تمرین: ساخت ساختار داده Stack(FIFO)

h.farhadi.py@gmail.com

Loop structures

h.farhadi.py@gmail.com

while

h.farhadi.py@gmail.com

while

حلقه while در پایتون برای تکرار اجرای یک بلوک کد
تا زمانی که یک شرط خاص برقرار باشد استفاده می‌شود.

```
while condition:
    # کدی که تا زمانی که شرط True باشد اجرا می‌شود
    # ...
```

```
count: int = 5
while count > 0:
    print(count)
    count -= 1

print(“Throw it”)
```

h.farhadi.py@gmail.com

تمرین: لاگین کردن (while, if, else)

h.farhadi.py@gmail.com

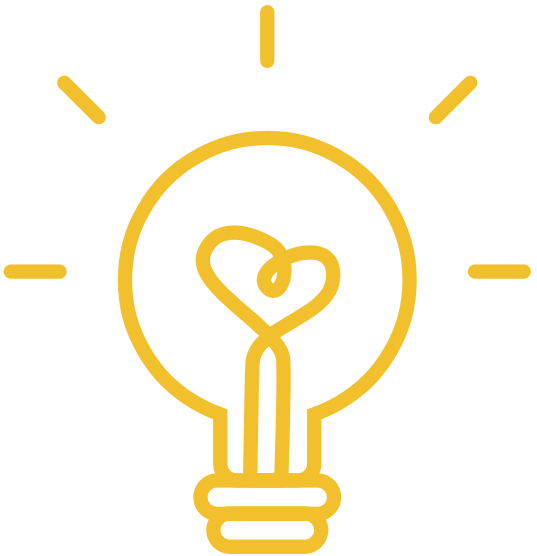
break

h.farhadi.py@gmail.com

break

دستور break برای خروج از loop است

```
count: int = 5

while count > 0:
    print(count)
    count -= 1

    If count == 3:
        break

print(“Throw it”)
```



h.farhadi.py@gmail.com

continue

h.farhadi.py@gmail.com

continue

برای رد شدن از تکرار فعلی است

```
count: int = 5

while count > 0:

    If count == 3:
        continue

    print(count)
    count -= 1

print(“Throw it”)
```

h.farhadi.py@gmail.com

for

h.farhadi.py@gmail.com

for loop

حلقه for در پایتون برای تکرار روی عناصر یک دنباله
(لیست، رشته، ....) یا دیگر شیءهای قابل تکرار استفاده می‌شود.

```
for variable in iterable:
    # code
```

```
fruits: list = ["apple", "banana", "orange", "grape"]
```

```
for fruit in fruits:
    print(fruit)
```

h.farhadi.py@gmail.com

range function

h.farhadi.py@gmail.com

range function

برای تولید دنباله‌ای از اعداد استفاده می‌شود.

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

```
numbers: any = range(5)

print(f"type of numbers is {type(numbers)}")
```

h.farhadi.py@gmail.com

While or For

برای تکرار تا زمانی که شرط برقرار است while

برای تکرار روی دنباله‌ها for

تعداد تکرارها ممکن است نامشخص باشد while

تعداد تکرارها معمولاً مشخص است for

نیاز به تعریف و به‌روزرسانی متغیر شمارنده while

از پیش تعریف شده بر اساس دنباله while

بیشتر بدانید: در سیستم عامل از while برای event loop استفاده می شود

h.farhadi.py@gmail.com

Nested for

h.farhadi.py@gmail.com

Nested for

حلقه‌های for تو در تو به معنی قرار دادن یک حلقه for داخل حلقه for دیگر است. این تکنیک برای پردازش ساختارهای داده دو بعدی، تولید ترکیبات و انجام محاسبات ماتریسی بسیار مفید است.

```
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i} × {j} = {i*j}", end="\t")
    print()
```

پیچیدگی زمانی: حلقه‌های تو در تو می‌توانند پیچیدگی زمانی $O(n^2)$ یا بیشتر داشته باشند، بنابراین برای داده‌های بزرگ باید با احتیاط استفاده شوند.

h.farhadi.py@gmail.com

تمرین: پالیندروم یک رشته ورودی

h.farhadi.py@gmail.com

تمرین: لود کردن یک عکس و نمایش اعداد RGB

h.farhadi.py@gmail.com

تابع function

h.farhadi.py@gmail.com

تابع یا function

توابع در پایتون بلوکهای کدی هستند که یک کار خاص را انجام میدهند
و میتوانند بارها در برنامه فراخوانی - call شوند.

```
def function_name(parameters: type) → return_type:
    # codes
    return value  # Optional


def say_hello_to_world():
    print("Hello, World!")

# call
say_hello_to_world()
```

h.farhadi.py@gmail.com

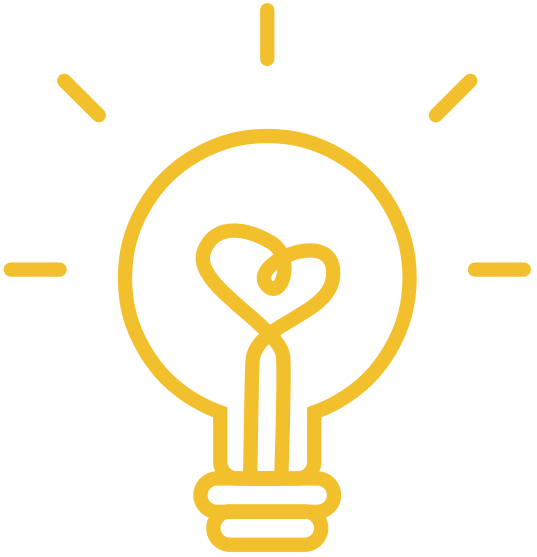
positional - پارامتر

```python
def greet(name: str, message: str) -> None:
    """
    a function that prints message + name
    with positional parameter
    """
    print(f"{message}, {name}")

# Call as Positional arguments, order is not important
greet("Hadi", "Hello")
greet("Hello", "Hadi")
# Call as Keyword arguments, order is not important
greet(message="Hello", name="Hadi")

# It gives us an error
greet("Hadi")
```

h.farhadi.py@gmail.com

default - پارامتر

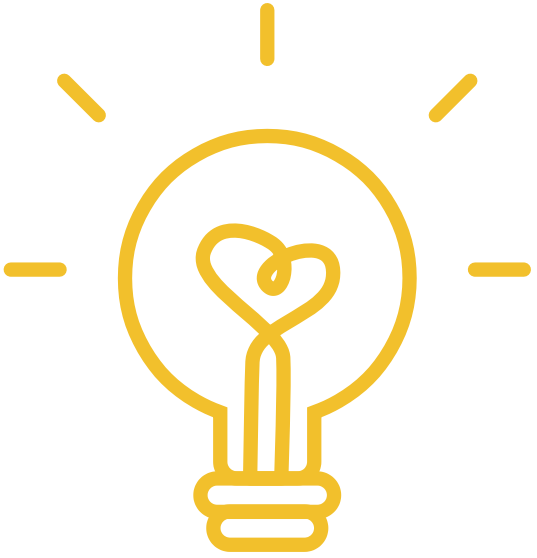
```python
def power(first_number: int, second_number: int = 2) -> int:
    power_result: int = first_number ** second_number
    return power_result
    # return first_number ** second_number

result: int = power(10)
print("power result = ", result)

result = power(10, 2)
print("power result = ", result)

result = power(10, 3)
print("power result = ", result)
```



h.farhadi.py@gmail.com

return

```python
def check_age(age: int) -> str:
    if age < 0:
        return "Invalid age"

    if age >= 18:
        return "Adult"
    else:
        return "Minor"


result = check_age(25)  # "Adult"
print(f"age result: {result}")
```



h.farhadi.py@gmail.com

Module

h.farhadi.py@gmail.com

Module

یک ماژول در پایتون فایلی با پسوند .py است که شامل کدهای پایتون (مانند توابع، کلاس‌ها، متغیرها و ...) است. ماژول‌ها به شما اجازه می‌دهند کدهای خود را سازماندهی کرده و از قابلیت استفاده مجدد (Reusability) بهره ببرید.

کاربردهای اصلی ماژول:

سازماندهی کد: تقسیم کد به بخش‌های منطقی و قابل مدیریت.

قابلیت استفاده مجدد: ذخیره کدها در ماژول‌ها و استفاده از آنها در پروژه‌های مختلف.

پنهان‌سازی اطلاعات: پیاده‌سازی جزئیات داخلی بدون تاثیرگذاری بر دیگر بخش‌های برنامه.

h.farhadi.py@gmail.com

Import Module

```
# math_operations.py

def add(first_number: int, second_number: int) -> int:
    return first_number + second_number

def subtract(first_number: int, second_number: int) -> int:
    return first_number - second_number


# main.py

import math_operations

sum_result: int = math_operations.add(12, 13)
print(sum_result)

subtraction_result: int = math_operations.subtract(12, 13)
print(subtraction_result)
```

h.farhadi.py@gmail.com

package

یک پکیج دایرکتوری‌ای است که شامل چندین ماژول و یک فایل __init__.py می‌باشد.

```
my_package/
    __init__.py
    module1.py
    module2.py
```

فایل __init__.py:

نشان می‌دهد که دایرکتوری یک پکیج پایتون است

می‌تواند خالی باشد یا شامل کد مقداردهی اولیه

کنترل می‌کند چه چیزی هنگام import پکیج در دسترس باشد

h.farhadi.py@gmail.com

تمرین: ساخت یک ماشین حساب از طریق ماژول و تابع

h.farhadi.py@gmail.com

exception

h.farhadi.py@gmail.com

exception

به زبان ساده، استثنا (Exception) یک اتفاق غیرمنتظره یا خطایی است که در حین اجرای برنامه رخ می‌دهد و جریان عادی برنامه را مختل می‌کند.

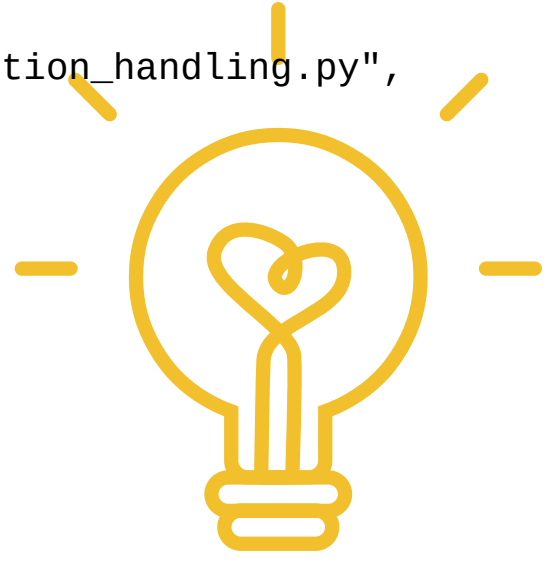
فرض کنید در حال رانندگی هستید (برنامه در حال اجراست).اگر بنزین تمام شود، یک "استثناء" رخ داده است. در این حالت:

ماشین متوقف می‌شود (برنامه crash می‌کند)

h.farhadi.py@gmail.com

```
first_number: int = int(input("Enter first number: ")) # 12
second_number: int = int(input("Enter second number: ")) # 0

result = first_number / second_number

print(f"{first_number} / {second_number} = {result}")


# ouput

Traceback (most recent call last):
  File "C:\Users\HadiFarhadi\PycharmProjects\PythonProject\exception_handling.py",
line 4, in <module>
    result = first_number / second_number
             ~~~~~~~~~~~~~^~~~~~~~~~~~~~~
ZeroDivisionError: division by zero

Process finished with exit code 1
```

ما نباید این خطا ها رو در نظر نگیریم و بگوییم: آخر چه کسی این تقسیم را انجام می دهد



h.farhadi.py@gmail.com

exception

چگونه تشخیص دهیم که چه exception احتمال دارد رخ دهد؟

در مستندات گفته می شود یا در docstring ها به آن اشاره می شود

و در مواقعی هم، که تعداد آن هم کم نیست هنگامی که برنامه به مشکل می خوردمتوجه این قضیه می شویم

exception – built in

ZeroDivisionError: تقسیم بر صفر

TypeError: عملیات روی نوع نادرست داده

ValueError: مقدار نامعتبر

FileNotFoundError: فایل پیدا نشد

IndexError: دسترسی به خارج از محدوده لیست

h.farhadi.py@gmail.com

exception handling – مدیریت استثناء

h.farhadi.py@gmail.com

مدیریت استثناء – exception handling

مدیریت استثناها در پایتون ابزاری قدرتمند برای کنترل خطاهای زمان اجرا (Runtime Errors) است

و به برنامه اجازه میدهد تا به صورت کنترل شده با شرایط غیرمنتظره برخورد کند. در ادامه به مفاهیم

کلیدی و نحوه استفاده از آن میپردازیم:

```
first_number: int = int(input("Enter first number: "))
second_number: int = int(input("Enter second number: "))

try:
    result: int = first_number / second_number
except ZeroDivisionError:
    print("You can't divide by zero")

# if ZeroDivisionError happened then the below code gives us an error,
another error!!!, how # can we solve this one? Scope?
print(result)

print(type(result))

print("Done")
```

اگر exception را مدیریت نکنیم برنامه متوقف می شود و کدهای دیگر اجرا نمی شوند. فرض

کنید یک سیستم فروش 24 ساعته داریم که در یک بخش کوچک آن این مشکل پیش آمده است

آیا کل برنامه باید متوقف شود؟

h.farhadi.py@gmail.com

نکته کلی، یا تمام کدهای داخل try بدون خطا انجام می شود یا بعد از اتفاق افتادن یک exception

سریعاً اجرای کدهای داخل try متوقف و وارد except می شود

```
first_number: int = int(input("Enter first number: "))
second_number: int = int(input("Enter second number: "))

try:
    result: int = first_number / second_number
    print(result)
    print(type(result))
except ZeroDivisionError:
    print("You can't divide by zero")


print("Done")
```

h.farhadi.py@gmail.com

دستور finally:

بلاک کد مربوط به finally همیشه اجرا می شود

```
try:
    first_number: int = int(input("Enter first number: "))
    second_number: int = int(input("Enter second number: "))
    result: int = first_number + second_number
    is_success: bool = True # scope
except (TypeError, ValueError) as e:
    print(f"exception: {e}")
    is_success: bool = False # scope
finally:
    print(f"the process has finished {"successfully" if is_success else
"failed"}") # scope
```

h.farhadi.py@gmail.com

Raise Exception

پرتاب استثنا (Raise Exception) به معنی ایجاد عمدی یک خطا در برنامه است.

این کار زمانی انجام می‌شود که می‌خواهیم شرایط خاصی را مدیریت کنیم یا خطای خاصی را گزارش دهیم.

```
person: dict[str, any] = {
    "name": "Alex",
    "age": 22,
    "city": "San Jose",
    "gas_capacity": 32
}

print("Person".center(50, "-"))
for field in person:
    print(f"{field.capitalize()}: {person[field]}")

# "gas_capacity" in person, in this case, I mean dict, this operation search in
keys
if "gas_capacity" in person:
    raise ValueError("You can't set gas_capacity for a person type")
```

در اینجا دقیقاً همان کاری را کردیم که پایتون هنگام مواجه با تقسیم بر صفر انجام می دهد

این کار برای این است که به لایه های دیگر خبر دهیم که قانونی شکسته شده است. پس به سبک خودتان آن را مدیریت کنید

h.farhadi.py@gmail.com

file

h.farhadi.py@gmail.com

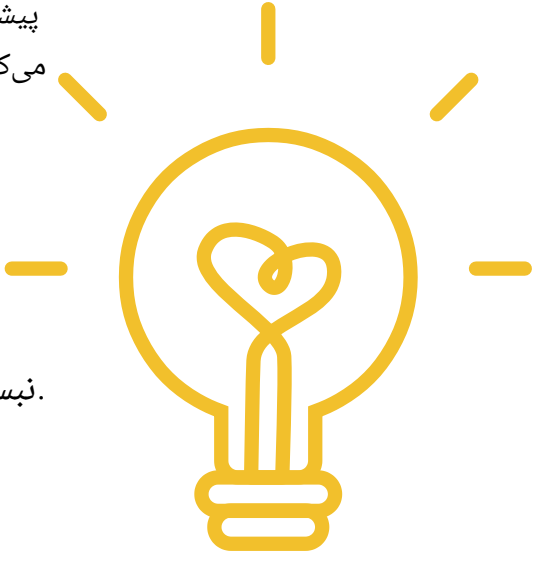
کار با فایل‌ها یکی از اساسی‌ترین مهارت‌ها در برنامه‌نویسی پایتون است.

باز کردن و بستن فایل

```
# users.txt

1   Hadi Farhadi    M   38
2   Ali Behnami M   32
3   Leila Rabbani   F   31
4   Azita Mahjoob   F   19
5   Ziba Kheradmand F   24
```

پیشوند r قبل از رشته در پایتون به معنای Raw String (رشته خام) است. این نوع رشته‌ها، کاراکترهای بک‌اسلش (\) را به صورت معمولی تفسیر می‌کنند و به آنها به عنوان کاراکترهای escape نگاه نمی‌کنند.

```
file_name: str = r'users.txt'

file = open(file_name, 'r', encoding='utf-8')
print("Users".center(50, '-'))

print(file.read())

file.close()
```



نبستن فایل در انتهای کار، منجر به نشست حافظه یا قفل شدن فایل شود.

h.farhadi.py@gmail.com

حالت های باز کردن فایل (mode)

‘r’

Read-only. Raises I/O error if file doesn't exist.

‘r+’

Read and write. Raises I/O error if the file does not exist.

‘w’

Write-only. Overwrites file if it exists, else creates a new one.

‘w+’

Read and write. Overwrites file or creates new one.

‘a’

Append-only. Adds data to end. Creates file if it doesn't exist.

‘a+’

Read and append. Pointer at end. Creates file if it doesn't exist.

‘rb’

Read in binary mode. File must exist.

‘rb+’

Read and write in binary mode. File must exist.

‘wb’

Write in binary. Overwrites or creates new.

‘wb+’

Read and write in binary. Overwrites or creates new.

‘ab’

Append in binary. Creates file if not exist.

‘ab+’

Read and append in binary. Creates file if it does not exist.

h.farhadi.py@gmail.com

with

استفاده از with (context manager) بهترین روش برای کار با فایل‌ها در پایتون است،

زیرا به صورت خودکار فایل را می‌بندد و از بروز خطاهای مربوط به باز ماندن فایل‌ها جلوگیری می‌کند.

```
try:
    with open('test.txt', 'r', encoding='utf-8') as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    print("file not found")
```

```
# read files line by line
try:
    with open('test.txt', 'r', encoding='utf-8') as file:
        for line in file:
            print(line.strip())
except FileNotFoundError:
    print('File not found')
```

h.farhadi.py@gmail.com

نوع tuple

h.farhadi.py@gmail.com

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.

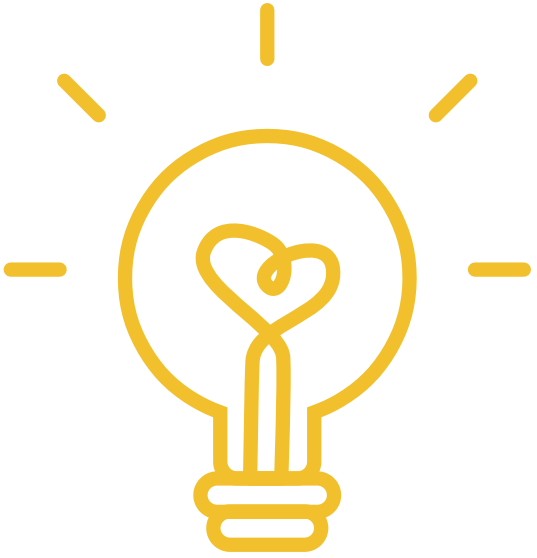
سریعتر از لیست است و حافظه ی کمتری مصرف می کند.

```
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))
```

h.farhadi.py@gmail.com

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]}")
```



h.farhadi.py@gmail.com

```python
# 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.

h.farhadi.py@gmail.com

نوع dict

h.farhadi.py@gmail.com

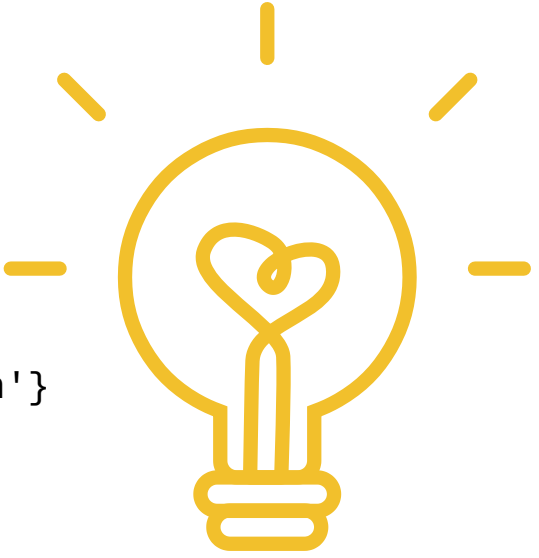
نوع dict

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.

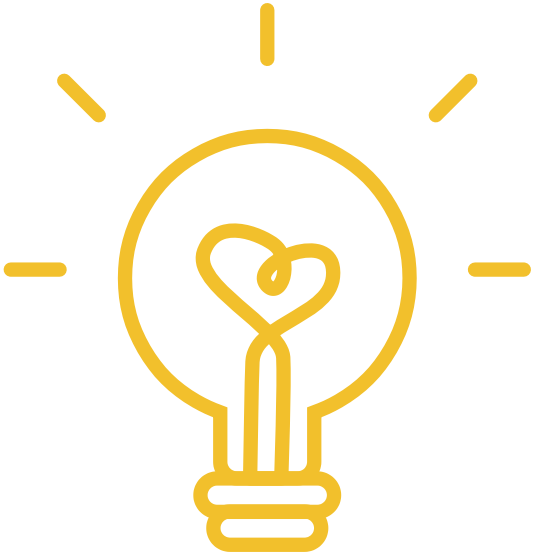
```
empty_dict: dict = {}
print(empty_dict)  # output: {}



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

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



h.farhadi.py@gmail.com

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")
```



h.farhadi.py@gmail.com

نوع set

h.farhadi.py@gmail.com

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}
```

h.farhadi.py@gmail.com

```
# 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)
```

h.farhadi.py@gmail.com

Call by value, Call by reference

h.farhadi.py@gmail.com

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)

h.farhadi.py@gmail.com

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)

h.farhadi.py@gmail.com

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
```

h.farhadi.py@gmail.com

Generators

h.farhadi.py@gmail.com

Generators

Generators (مولدها) یکی از قدرتمندترین ویژگی‌های پایتون هستند که به شما امکان می‌دهند iteratorهای سفارشی ایجاد کنید بدون اینکه نیاز به پیاده‌سازی کامل کلاس iterator داشته باشید.

Generator نوع خاصی از تابع است که به جای return از yield استفاده می‌کند و حالت (state) خود را بین فراخوانی‌ها حفظ می‌کند.

h.farhadi.py@gmail.com

Generators

```
number_list = [1, 2, 3, 4, 5, 6]

def get_number():
    index = 0
    number = number_list[index]
    index += 1
    return number

print("Default function".center(50, '-'))
print(get_number()) # output: 1
print(get_number()) # output: 1
print(get_number()) # output: 1


def get_number_with_yield():
    index = 0
    number = number_list[index]
    index += 1
    yield number
    number = number_list[index]
    index += 1
    yield number
    number = number_list[index]
    index += 1
    yield number


print("Generator function".center(50, '-'))
number_generator = get_number_with_yield()
print(next(number_generator)) # output: 1
print(next(number_generator)) # output: 2
print(next(number_generator)) # output: 3
```

h.farhadi.py@gmail.com

Generators

```
# مشابه list comprehension اما با پرانتز
generator = (x**2 for x in range(5)) # ساخت یک generator بدون استفاده از تابع

print(list(generator))  # [0, 1, 4, 9, 16] برای داده های بزرگ این عمل خطرناک است
```

Generator برای داده های خیلی بزرگ مناسب هستند چرا که مدیریت حافظه (رم) عالی دارند. چون در هر لحظه به یک عنصر دسترسی داریم next. از جابجایی بین رم و هارد استفاده می کند. پس احتمال پر شدن رم برای داده ای بزرگ به شدت پایین و در حد صفر است. با این اوصاف سرعت کمتری نسبت به لیست داشته باشد

h.farhadi.py@gmail.com

decorators

h.farhadi.py@gmail.com

Decorators

Decorators (دکوراتورها) یکی از قدرتمندترین و پیشرفته‌ترین ویژگی‌های پایتون هستند که به شما امکان می‌دهند رفتار توابع یا کلاس‌ها را بدون تغییر کد اصلی آن‌ها تغییر دهید.

Decorator یک تابع است که تابع دیگری را به عنوان ورودی می‌گیرد و عملکرد جدیدی به آن اضافه می‌کند.

```
@decorator
def function():
   Pass
```

```
def function():
    pass

function = decorator(function) # معادل @
```

h.farhadi.py@gmail.com

Decorators

```
def simple_decorator(func):
    def wrapper():
        print("قبل از فراخوانی تابع")
        func()
        print("بعد از فراخوانی تابع")
    return wrapper


def say_hello():
    print("Hello!")


func = simple_decorator(say_hello)
func()
```

h.farhadi.py@gmail.com

Decorators

```
def simple_decorator(func):
    def wrapper():
        print("قبل از فراخوانی تابع")
        func()
        print("بعد از فراخوانی تابع")
    return wrapper


@simple_decorator
def say_hello():
    print("Hello!")


say_hello()
```

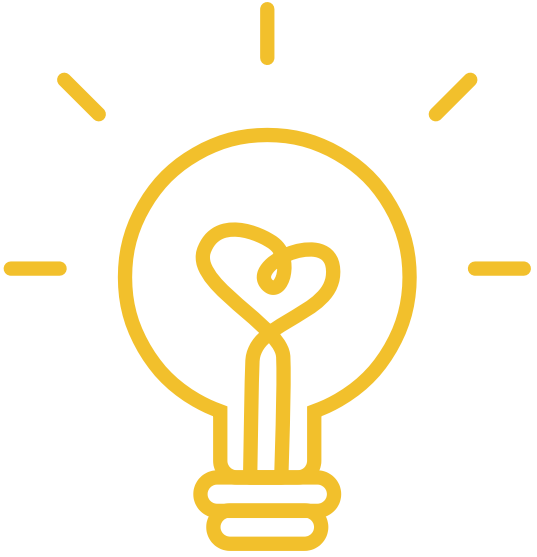
Flask, FastAPI به وفور و django براساس نیاز از دکوریتور استفاده می کند

h.farhadi.py@gmail.com

Decorators

#دکوریتوری که تابع آن ورودی و خروجی دارد

```
def decorator_with_args(func):
    def wrapper(*args, **kwargs):
        print(f"تابع {func.__name__} با آرگومان‌ها فراخوانی شد")
        print(f"آرگومان‌ها: {args}, {kwargs}")
        result = func(*args, **kwargs)
        print(f"نتیجه: {result}")
        return result
    return wrapper

@decorator_with_args
def add(a, b):
    return a + b

result = add(5, 3)
```



h.farhadi.py@gmail.com

توابع بازگشتی (Recursive Functions)

h.farhadi.py@gmail.com

توابع بازگشتی (Recursive Functions)

توابع بازگشتی توابعی هستند که خود را فراخوانی می‌کنند تا یک مسئله را به مسائل کوچکتر تقسیم کنند. این توابع برای حل مسائلی که می‌توانند به زیرمسائل مشابه تقسیم شوند بسیار مناسب هستند.

شرط پایه (Base Case): شرطی که بازگشت را متوقف می‌کند. همیشه شرط پایه داشته باشید: بدون شرط پایه، تابع بی‌نهایت اجرا می‌شود

گام بازگشتی (Recursive Step): فراخوانی تابع با ورودی کوچکتر

h.farhadi.py@gmail.com

توابع بازگشتی (Recursive Functions)

```
def factorial(number: int) -> int:
    # Base Case
    if number == 0 or number == 1:
        return 1
    # Recursive Step
    else:
        return number * factorial(number - 1)

print(factorial(6))  # 120
```

h.farhadi.py@gmail.com

Functional Programming

h.farhadi.py@gmail.com

برنامه‌نویسی تابعی یک پارادایم برنامه‌نویسی است که در آن از توابع خالص (Pure Functions) استفاده می‌شود و از تغییر حالت (State) و داده‌های mutable اجتناب می‌شود.

```
# ایجاد متغیر جدید به جای تغییر متغیر اصلی
# incorrect
numbers = ["Dembele", "Yamal", "Salah"]
numbers.append("Rafinia")  # change the main data

# correct
numbers = ["Dembele", "Yamal", "Salah"]
new_numbers = numbers + ["Rafinia"]  # create a new list without reference to the main data

new_numbers.append("Salah")

print(numbers)
print(new_numbers)
```

مصرف حافظه بیشتر

h.farhadi.py@gmail.com

توابعی که توابع دیگر را به عنوان ورودی می‌گیرند یا برمی‌گردانند

```
def apply_function(func, number):
    """run another function and return the result"""
    return func(number)

def square(number):
    return number * number

result = apply_function(square, 5)  # 25
```

h.farhadi.py@gmail.com

تمرین: ساخت فروشگاه آنلاین خط فرمان(تمامی عملیات اعتبارسنجی کاربر و گرفتن کالا ها باید از طریق فایل json انجام شود) اما عملیات اضافه کردن کالا به سبد خرید در رم ذخیره شود.

h.farhadi.py@gmail.com

THANK YOU

h.farhadi.py@gmail.com