اصول مقدماتی پایتون 4

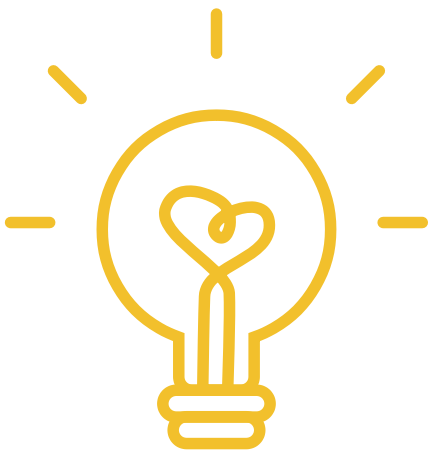
files, exceptions

هادی فرهادی

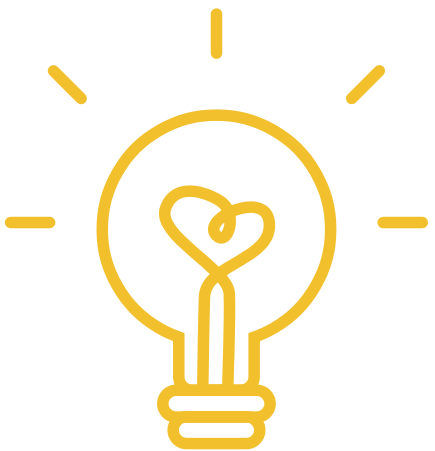
شهریور ۱۴۰۴

h.farhadi.py@gmail.com

یادآوری



سؤال جواب



exception

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.

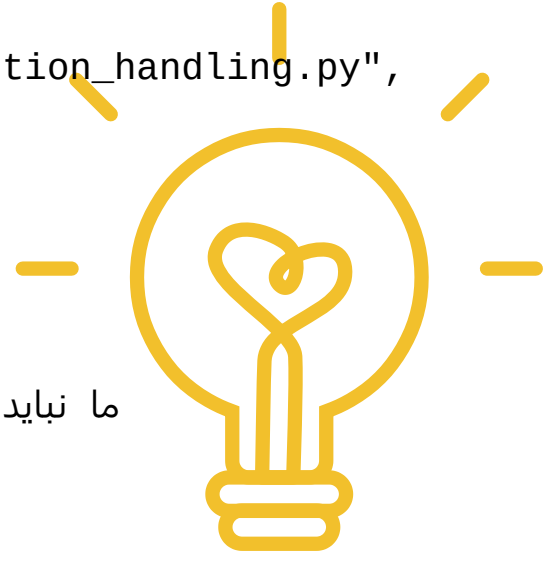
فرض کنید در حال رانندگی هستید (برنامه در حال اجراست).اگر بنزین تمام شود، یک "استثناء" رخ داده است. در این حالت:

ماشین متوقف می‌شود (برنامه crash می‌کند)

exception

```
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



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

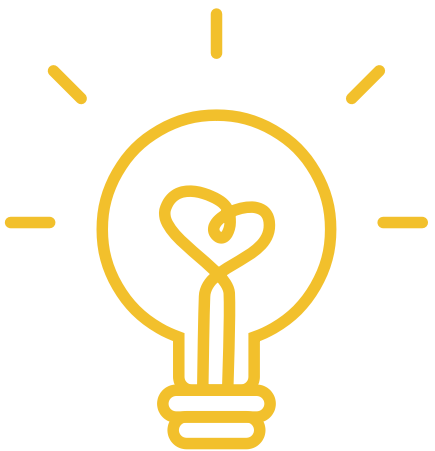
TypeError: عملیات روی نوع نادرست داده

ValueError: مقدار نامعتبر

FileNotFoundError: فایل پیدا نشد

IndexError: دسترسی به خارج از محدوده لیست

مدیریت استثناء – exception handling



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؟

مدیریت استثناء – exception handling

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")
```

مدیریت استثناء – exception handling

هر کدی که کار نکرد یا خطا داد را نباید با try -except حل کنید

کد زیر کاملاً اشتباه است

```
try:
    xxxxxx
except:
    print("invalid characters")
```

وقتی خطایی رخ داد حتماً باید اطلاع رسانی انجام شود که مدیران در جریان قرار بگیرند

فرض کنید که قطعه کدی برای کاربران ایمیل ارسال می کند. وقتی این قطعه کد اجرا می شود

میل سرور متوقف شده است. در این حالت بایستی اطلاع رسانی در سریعترین زمان انجام شود

تا مشکل را رفع کنند. اما وقتی کاربر برای یک تقسیم بر صفر انجام داده است باید به خود مشتری یا کاربر

بگوییم که نباید تقسیم بر صفر انجام دهید

مدیریت استثناء – exception handling

except بدون مشخص کردن نوع خطا (جذب همه خطاها)

از این روش استفاده نکنید از آنجایی که همه خطا ها را می گیرد

متوجه مشکل اصلی برنامه نمی شویم.

کدهای بعدی نمی‌دانند چه خطاهایی ممکن است رخ دهد

و فهم برنامه سخت تر 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
except:
    print("You can't divide by zero")


print(result)

print(type(result))

print("Done")
```

مدیریت استثناء – exception handling

except بدون مشخص کردن نوع خطا (جذب همه خطاها)

از این روش استفاده نکنید از آنجایی که همه خطا ها را می گیرد

متوجه مشکل اصلی برنامه نمی شویم.

کدهای بعدی نمی‌دانند چه خطاهایی ممکن است رخ دهد

و فهم برنامه سخت تر 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
except:
    print("You can't divide by zero")


print(result)

print(type(result))

print("Done")
```

مدیریت استثناء – exception handling

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
```

استفاده از else (اگر خطایی رخ ندهد)

```
try:
    first_number: int = int(input("Enter first number: "))
    second_number: int = int(input("Enter second number: "))
    result: int = first_number + second_number
except (TypeError, ValueError) as e:
    print(f"exception: {e}")
else:
    print(f"result: {result}")
```

مدیریت استثناء – exception handling

```
def operate():
    try:
        first_number = int(input("Enter first number: "))
        second_number = int(input("Enter second number: "))
        result: int = first_number / second_number
        return result
    except ZeroDivisionError:
        print("Division by zero")
    except TypeError as error:
        print(f"TypeError: {error}")
    else: # execution ?
        print("Operation successful")
    finally:
        print("End") # execution ?

result = operate()
print(result)
```

Raise Exception

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

بیشتر بدانید - Separation of Concerns (جداسازی نگرانی‌ها)

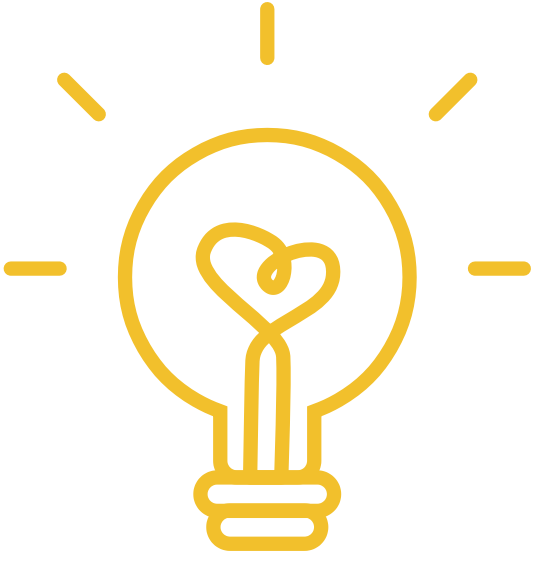
بیشتر بدانید - Separation of Concerns (جداسازی نگرانی‌ها)

Separation of Concerns یا SoC یک اصل مهم در مهندسی نرم‌افزار است که می‌گوید:

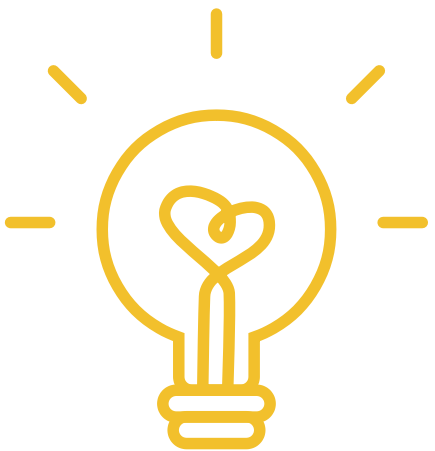
"یک برنامه باید به بخش‌های مجزا تقسیم شود، هر بخش مسئولیت جداگانه‌ای داشته باشد و نگرانی خاصی را حل کند."

هر جزء سیستم باید: یک کار را انجام دهد، یک مسئولیت داشته باشد و یک نگرانی را حل کند. که برنامه را قابل نگهداری می کند. یعنی اعمال تغییرات در آینده راحت تر خواهد شد

```
📦 Application
├── 🧩 Controllers          # مدیریت درخواست‌ها
├── 🧩 Services             # منطق کسب‌وکار
├── 🧩 Repositories         # دسترسی به دیتابیس
├── 🧩 Models               # مدل‌های داده
├── 🧩 Validators           # اعتبارسنجی
└── 🧩 Utilities            # ابزارهای کمکی
```



Assertions



Assertions

دستور assert یک ابزار دیباگینگ و اعتبارسنجی در پایتون است که برای بررسی

شرایطی که باید همیشه درست باشند استفاده می‌شود. اگر شرط نادرست باشد،

پایتون یک استثنای AssertionError ایجاد می‌کند.

```
def divide(first_number, second_number):
    assert second_number != 0, "division by zero"
    return first_number / second_number

result = divide(10, 2)
result = divide(10, 0)  # AssertionError: division by zero
```

```
def process_string(text):
    assert isinstance(text, str), "value is not a string(str)"
    return text.upper()

result: str = process_string("hello")

print(result) # output: HELLO
process_string(123)     # AssertionError: value is not a string(str)
```

file

file

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.

file

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.

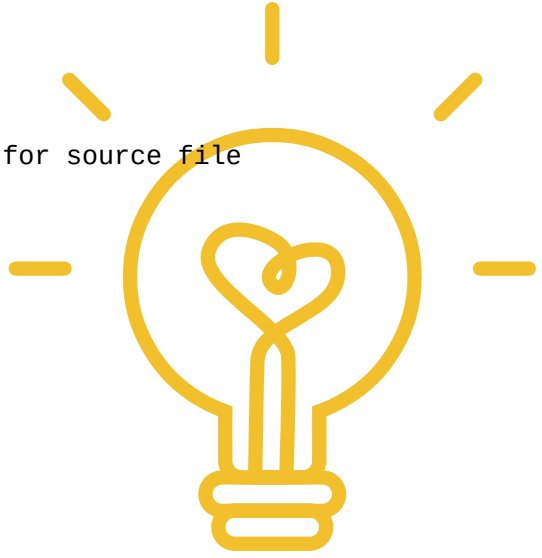
‘ab+’

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

file

مثال

```
# Copying a file

source_file = None
dest_file = None

try:
    source_file = open('source.txt', 'r', encoding='utf-8')
    dest_file = open('copy.txt', 'w', encoding='utf-8')

    content = source_file.read()
    dest_file.write(content)

    print('copying done')

except FileNotFoundError:
    print('The source file was not found.') # TWO ++ gift: why do we only show this error for source file
except IOError as e:
    print(f'I/O Error: {e}')
finally:
    if source_file:
        source_file.close()
    if dest_file:
        dest_file.close()
```



file

مثال

```
import datetime

file = None

try:
    file = open('log.txt', 'a', encoding='utf-8')


    now = datetime.datetime.now()
    file.write(f'login time: {now}\n')
    print('Your login time store successfully!')

except IOError as e:
    print(f'writing to file error: {e}')
finally:
    if file:
        file.close()
```

file

مثال

```
# write to file
try:
    file = open('test.txt', 'w', encoding='utf-8')
    file.write('سلام دنیا!\n')
    file.write('این یک متن تست است.\n')
    file.write('خط سوم فایل.\n')
except FileNotFoundError:
    print("file not found")
finally:
    if file:
        file.close()  # this is very important
```

```
# reading line by line from file
file = open('test.txt', 'r', encoding='utf-8')

try:
    print('reading line by line:')
    line = file.readline()
    while line:
        print(line.strip())  # remove \n, space, tab
        line = file.readline()
finally:
    file.close()
```

file

```
file = open('test.txt', 'r')
content = file.read()
print(content)
# very dangerous: if an error happened here, file won’t close anymore
file.close()
```

وقتی می گویم این کار خیلی خطرناک است منظور وقتی است

که کد یک شرکت بالا (یعنی روی اینترنت است یا دست مشتری است) است

بنابراین یا داخل try-except-finally استفاده شود یا از with استفاده شود

with

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')
```

with

```
# writing to a file
try:
    with open('output.txt', 'w') as file:
        file.write('سلام دنیا!\n')
        file.write('این یک متن فارسی است.\n')
except FileNotFoundError:
    print('File not found')
```

کد فوق یک exception دارد

```
# append to a file
with open('output.txt', 'a', encoding='utf-8') as file:
    file.write('این خط به انتهای فایل اضافه می‌شود.\n')
```

with

```
# write multiple lines to a file
try:
    lines = ['خط اول\n', 'خط دوم\n', 'خط سوم\n']
    with open('output.txt', 'w', encoding='utf-8') as file:
        file.writelines(lines)
except FileNotFoundError:
    print('File not found')
```

```
try:
    with open('simpson.png', 'rb') as source:
        with open('simpson_copy.jpg', 'wb') as destination:
            destination.write(source.read())
except FileNotFoundError:
    print('File not found')
```

with

```
import csv

# write to csv file
with open('data.csv', 'w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerow(['نام' ,'سن' ,'شهر'])
    writer.writerow(['علی' ,'25' ,'تهران'])
    writer.writerow(['سارا' ,'30' ,'شیراز'])
```

```
# read from a csv file
with open('data.csv', 'r', encoding='utf-8') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

with

```
import json

data = {
    'نام' : 'علی',
    'سن' : 25,
    'شهر' : 'تهران'
}

# write to json file
with open('data.json', 'w', encoding='utf-8') as file:
    json.dump(data, file, ensure_ascii=False, indent=4)
```

```
# read from json file
with open('data.json', 'r', encoding='utf-8') as file:
    loaded_data = json.load(file)
    print(loaded_data)
```

with

```
# python 3.10+ multiple files, copying a file
with (
    open('items.txt', 'r', encoding='utf-8') as source,
    open('destination.txt', 'w', encoding='utf-8') as dest
):
    content = source.read()
    dest.write(content)
```

Data Encoding & Number Systems

سیستم اعداد دسیمال (Decimal) - مبنای ۱۰

اعداد ۰ تا ۹

سیستم مورد استفاده در زندگی روزمره

مثال: $10^0 \times 3 + 10^1 \times 2 + 10^2 \times 1 = 123$

سیستم اعداد باینری (Binary) - مبنای ۲

اعداد ۰ و ۱

زبان اصلی کامپیوترها

مثال: $1011_2 = 2^3\times1 + 2^2\times0 + 2^1\times1 + 2^0\times1 = 11_{10}$

کوچکترین واحد حافظه بیت است صفر یا یک

به هر هشت بیت یک بایت می گویند.

کیلو بایت

مگابایت

گیگابایت

سیستم اعداد هگزادسیمال (Hexadecimal) - مبنای ۱۶

اعداد ۰-۹ و حروف A-F

نمایش فشرده باینری

مثال: $A3F_{16} = 10\times16^2 + 3\times16^1 + 15\times16^0 = 2623_{10}$

سیستم اعداد اکتال (Octal) - مبنای ۸

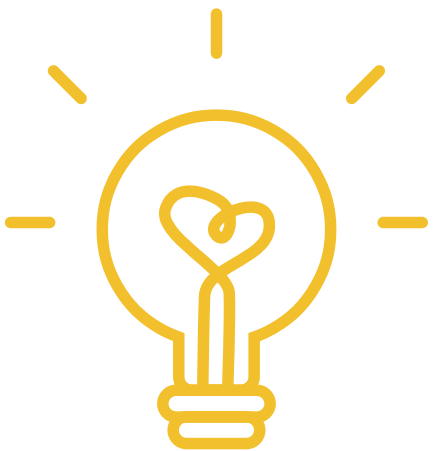
اعداد ۰ تا ۷

کمتر رایج اما در برخی سیستم‌ها استفاده می‌شود

تبدیل بین سیستم اعداد

```
# binary representation of 11
binary_number: bin = 0b1011

print(binary_number) # output: 11



# hexadecimal representation 255
hex_number: hex = 0xFF

print(hex_number) # output: 255


number: int = 345

print(f"hex of {number} is {hex(number)}")
print(f"binary of {number} is {bin(number)}")
```

انکودینگ داده (Data Encoding)



ASCII

انکودینگ (Encoding): تبدیل داده از یک فرمت به فرمت دیگر برای مواردی مانند:

ذخیره‌سازی کارآمد، انتقال داده، امنیت داده، سازگاری بین سیستم‌ها

دیکودینگ (Decoding): بازگرداندن داده انکودشده به فرمت اصلی آن

ASCII (American Standard Code for Information Interchange)

128 کاراکتر (7 بیت)

کاراکترهای انگلیسی، اعداد، و نمادها

```
# Convert to ASCII
text = "Hello"
ascii_values = [ord(char) for char in text]
print(ascii_values)  # [72, 101, 108, 108, 111]

# Convert From ASCII
ascii_list = [72, 101, 108, 108, 111]
text = ''.join([chr(code) for code in ascii_list])
print(text)  # Hello
```

Unicode (UTF-8, UTF-16, UTF-32)

Unicode (UTF-8, UTF-16, UTF-32)

پشتیبانی از تمام زبان‌های جهان

UTF-8: متغیر طول (1-4 بایت per character)

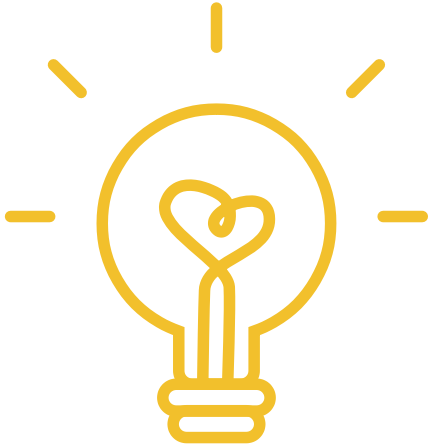
```
# encoding
text = "سلام دنیا!"
utf8_encoded = text.encode('utf-8')
print(utf8_encoded)  # b'\xd8\xb3\xd9\x84\xd8\xa7\xd9\x85
\xd8\xaf\xd9\x86\xdb\x8c\xd8\xa7!'

# decoding
decoded_text = utf8_encoded.decode('utf-8')
print(decoded_text)  # سلام دنیا!
```

مثال

```
def hex_to_rgb(hex_color):
    hex_color = hex_color.lstrip('#')
    return tuple(int(hex_color[i:i+2], 16) for i in (0, 2, 4))

def rgb_to_hex(rgb):
    return '#{:02x}{:02x}{:02x}'.format(*rgb)

print(hex_to_rgb("#ff0000"))  # (255, 0, 0)
print(rgb_to_hex((255, 0, 0)))  # #ff0000
```

این تبدیل مخصوص گرافیست ها است

Bitwise Operators - بیشتر بدانید



Bitwise Operators - بیشتر بدانید

عملگرهای بیتی برای انجام عملیات مستقیم روی بیت‌هایاعداد استفاده می‌شوند.

این عملگرها محاسبات را در سطح بیتی انجام می‌دهند.

& AND اگر هر دو بیت 1 باشند، نتیجه 1 است

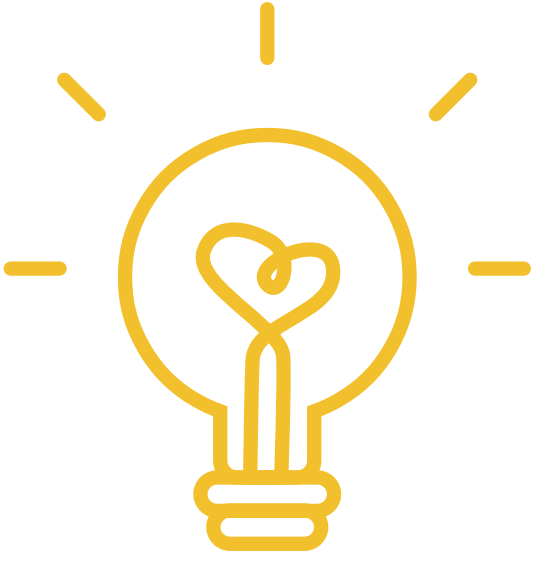
| OR اگر حداقل یکی از بیت‌ها 1 باشد، نتیجه 1 است

^ XOR اگر بیت‌ها متفاوت باشند، نتیجه 1 است

~ NOT معکوس کردن تمام بیت‌ها

<< Left Shift جابجایی بیت‌ها به چپ

>> Right Shift جابجایی بیت‌ها به راست



Bitwise Operators - بیشتر بدانید

```
first_number: int = 12  # 1100
second_number: int = 10  # 1010

result = first_number & second_number
print(f"{first_number} in binary is {bin(first_number)}")
print(f"{second_number} in binary is {bin(second_number)}")
print(f"{bin(first_number)} & {bin(second_number)} = {bin(result)}, {result}")
```

```
first_number: int = 5  # 101 در باینری
Result: int = first_number << 2
print(result)  # 20 :خروجی
# توضیح:
# 101 (5) با دو shift (20) 10100 :به چپ
# 20 = 2^2 * 5 :معادل
```

bytes, bytearray - بیشتر بدانید

بیشتر بدانید - bytes, bytearray

در پایتون، bytes و bytearray برای کار با داده‌های باینری و مدیریت مستقیم حافظه استفاده می‌شوند.

این نوع داده‌ها برای کار با فایل‌های باینری، پروتکل‌های شبکه، رمزنگاری و سایر عملیات سطح پایین ضروری هستند.

bytes غیرقابل تغییر (immutable) هستند

bytearray قابل تغییر (mutable) هستند

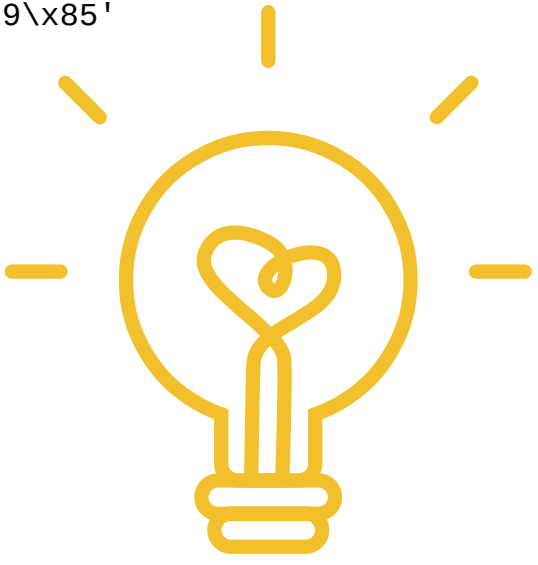
```
# list of numbers
bytes_number_list = bytes([65, 66, 67, 68])  # b'ABCD'
print(bytes_number_list)

# encoding
bytes_str = bytes("سلام", 'utf-8') #
b'\xd8\xb3\xd9\x84\xd8\xa7\xd9\x85'
print(bytes_str)

#  literal
bytes_literal_str = b"Hello"  # b'Hello'
print(bytes_literal_str)
```

bytes, bytearray - بیشتر بدانید

```
# from a number list
bytearray_number_list = bytearray([65, 66, 67, 68])  #
bytearray(b'ABCD')
print(bytearray_number_list)

# encoding
bytearray_utf_str = bytearray("سلام", 'utf-8')
print(bytearray_utf_str)

# literal
bytearray_literal_str = bytearray(b"Hello")
print(bytearray_literal_str)
```

bytes, bytearray - بیشتر بدانید

```
# str to bytes
text = "Hello World"
byte_data = text.encode('utf-8')  # b'Hello World'
print(byte_data)

# bytes to str
decoded_text = byte_data.decode('utf-8')  # 'Hello World'
print(decoded_text)

# persian text
persian_text = "سلام"
persian_bytes = persian_text.encode('utf-8')  # b'\xd8\xb3\xd9\x84\xd8\xa7\xd9\x85'
print(persian_bytes)
back_to_text = persian_bytes.decode('utf-8')  # 'سلام'
print(back_to_text)
```



bytes, bytearray - بیشتر بدانید

```
# bytes to list
data = b"ABC"
number_list = list(data)  # [65, 66, 67]
print(number_list)

# to bytes
new_bytes = bytes(number_list)  # b'ABC'
print(new_bytes)

# from bytearray
new_bytearray = bytearray(number_list)  # bytearray(b'ABC')
print(new_bytearray)
```

THANK YOU

h.farhadi.py@gmail.com