Курс: Python 3

Python е един от най-гъвкавите и мощни езици в момента, използван в web, computer science, big data, протипизиране, управление на устройства, computer vision, machine learning или просто за забавление. Python е изключително лесен за научаване и използване, а мощните му възможности и хилядите външни библиотеки правят работата с него удоволствие. Познанията в този език са MUST умение за всеки един програмист на всяко едно ниво.

**Обща информация за курса**

Този курс има за цел да ви запознае с езика Python, неговата философия, която често се нарича “The pythonic way” като за целта ще се използват множество задачи от реалния живот и визуални примери - учене чрез практика и забавление. В този курс ще начуите всичко необходимо, за да решавате реални задачи с Python, като използвате правилните езикови конструкции.

Всички лекции ще са с продължителност 4 астрономически часа. По време на лекциите основно ще се пише код върху реални, практически задачи.

**За кого е подходящ курса?**

Курсът е насочен както за начинаещи, така и за малко по-напреднали в програмирането. Не е необходимо да имате предишен опит с Python, но е силно препоръчително да имате обща култура в програмирането и разбирането на променливи, функции, логически и условни оператори.

**Какви умения ще придобия?**

Създаване на чист и ясен програмен код - “The Pythonic way”, които ще Ви бъдат от полза и при работа с други езици Разбиране на езика Свободна работа със стандартната библиотека Използване на външни библиотеки Работа със структури от данни Обектно ориентирано програмиране в Python Работа с файлове, стандартни формати (CSV, JSON, XML), мрежа и бази данни Познаване на особеностите на работата с време и дати - часови зони, високосни години, календари и всичко, което е свързано с работа с времева информация.

# 1. Въведение в Python 3

## Курсът

Курсът, който ние бихме искали да изкараме :о)

По време на този курс ще се постараем по забавен и практически ориентиран подход да Ви запознаем с **Python 3** - един лесен, но изключително мощен език за програмиране. Освен за самия език, ще обясним и доста важни, общовалидни концепции, които са в основата на съвременното програмиране.

Осигурете си надеждна машина, тъй като ще пишем доста код. Доста ще Ви е полезно да следвате и живите примери, които ще показваме по време на лекциите.

Особено в началото, някои неща ще бъдат “щото така”, за да не се отклоняваме от темата. С течение на курса ще ги обясним в подробности.

**Пазете си решенията на задачите** - често ще се връщаме към едни и същи задачи, които ще доразвиваме така, че да се приближават все повече до реалните ситуации и условия, които може срещнете в работата си.

##### **Какво няма да има в този курс?**

* **“Hello world”** - всеки път, когато на презентация и лекция видите някой да показва "Hello world", това означава, че човекът пред Вас или го е домързяло, или не може да измисли по-смислено приложение на демонстрираната технология. Ще направим всичко възможно в този курс да няма "Hello world" - даже и най-простия демонстрационен код би могъл да върши някаква полезна работа :о)
* Web с Python - разработката на web приложения с Python e доста широка, и ще бъде тема за отделен курс.

### "Таковата"

**"Таковата"** е нашата нова и експериментална система, с която ще се опитаме да направим курса малко по-динамичен, и доста по-полезен.

В **"Таковата"** ще качваме основните моменти от лекциите - структура на лекцията, важни описания и дефиниции, и линкове към ресурси по темата в Интернет. И въпреки, че не целим съдържанието на лекциите в "Таковата" да е съпоставимо със "Записки по българските възстания" - все пак повечето материал ще бъде под формата на живи демонстрации и задачи, определено според нас ще намалим ефектите "шрифта", "проектора", "последния ред" и "чакай да препиша линка" :о)

По-ценното в **Таковата** всъщност ще блесне, когато дойде време за задачите. Обикновено по време на подобни курсове задачите остават в края на лекцията, и това се прави по една основна причина - всеки решава задачите за различно време. Ако задачите са по средата на лекцията, или се губи много време докато всички решат задачата, или пък задачата се спира, а немалка част от хората все още не са приключили с решаването й.

От друга страна, когато задачите са в края, е въможно да се загуби връзката с преподавания материал.

Ние сме възприели подход, с който ще се опитаме да извлечем най-доброто и от двата метода. Задачите ще бъдат по време на лекцията - веднага, след като покажем или обясним нещо. За да не губим прекалено много време в чакане, ще преминаваме напред, когато **80% от присъстващите в залата успеят да решат задачата**.

Условията на задачите ще бъдат разписани в системата, но на страницата на задачата ще има и 3 магически бутона - "Готово","Проблем" и "Въпрос". Ако успеете да решите задачата, моля дайте ни знак с бутона "Готово". На лекторския екран това ще се отрази в реално време, и ще можем веднага да определим кога да продължаваме напред.

##### **Защо да си качвате решенията в Таковата?**

Освен магическите 3 бутона, можете да качвате и решенията в системата - полето за код се намира точно под условието на задачата.

Въпреки, че въобще не е задължително да го правите, качването на решенията в системата ще помогне много на всички ни, защото ще ни даде възможност:

* ... да прегледаме как се справяте на момента, без да е необходимо да Ви обикаляме като пчелички и да Ви разпитваме;
* ... да покажем добрите решения на дъската и да ги обсъдим заедно. Дали искате името Ви да се показва, когато показваме решението зависи изцяло от Вас (по подразбиране имената няма да бъдат показвани).
* ... да видим на момента често срещани грешки, което може да означава, че не сме обяснили достатъчно добре нещата по време на лекции, а това е една от най-важните обратни връзки за един лектор;
* ... да съхраните решенията си на още едно място.

## Кои сме ние

##### **Иван Ванков (gatakka)**

[gatakka@gatakka.eu](mailto:gatakka@gatakka.eu?subject=%22Python%203%20course%22)

##### **Борис Червенков**

С повече от 20 години професионален опит в информационните технологии, през които съм бил част от почти всички аспекти на IT индустрията - разработка на софтуер за множество платформи, технологии и езици за програмиране, обучение, управление на екипи и проекти, sales, UI/UX и quality assurance. Работя с Python вече повече от 7 години.

Можете да ме намерите в Twitter - [@b\_chervenkov](https://twitter.com/@b_chervenkov), или [boris.chervenkov@gmail.com](mailto:boris.chervenkov@gmail.com?subject=%22Python%203%20course%22)

## Философията на Python

>>> import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than \*right\* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

Eдин съдържателен post от създателя на Python - Guido van Rossum

[The History of Python: Python's Design Philosophy](http://python-history.blogspot.bg/2009/01/pythons-design-philosophy.html)

##### **Чист и ясен програмен код**

В реалнността, огромна част от живота на един проект всъщност е поддръжката и развитието му.

Рядкост са програмните езици, които толкова много държат на четимостта на кода, както Python. Самият факт, че идентацията е в основата на структурирането на кода, е изключително показателен за това. В Python не само че трудно ще видите неподреден код, но най-вероятно той няма и да работи:

#include <stdio.h>

int main(void) {

int sum = 0;

int current;

do {

printf("\nEnter an integer > ");

scanf("%d", &current);

if (current > 0) sum = sum + current;

} while (current > 0);

printf("\nThe sum is %d\n", sum);

}

Именованите параметри, краткия запис, липсата на [cruft](https://en.wikipedia.org/wiki/Cruft), както и много други аспекти на Python, които ще разгледаме по време на курса, правят кода по-лесен за четене и разбиране.

В Python общността е възприет стил на писане, описан в т.нар. [PEP8](https://www.python.org/dev/peps/pep-0008/) . Най-основните правила, към които ще искаме да се придържаме, включват:

* идентацията на кода е с интервали (не с tab-ове);
* най-често идентацията е 4 интервала;
* имената на променливите и функциите са с т.нар. "snake\_case" - малки букви, и разделител долно тире ("подчертавка") между отделните думи;
* имената на класовете са т.нар. "CamelCase" - както при C++, Java & C#;
* трябва да има интервали около операторите (напр. if a == b: ...., а не if a==b: ...), като изключение са само именованите параметри на функции и методи, които ще разгледаме в следващите лекции.
* трябва да има поне 2 празни реда между дефиниции на функции и класове;

##### **"Batteries included"**

Стандартната библиотека на Python включва немалко модули и пакети, които ще Ви помогнат в решаването на редица проблеми и задачи.

Например, в Python има прост web сървър, с който можете лесно и бързо да тествате или да споделите директория, даже и да не сте програмист на Python:

$ python3 -m http.server

## Особености на синтаксиса

##### **"Kъде са ми скобичките?"**

..., пита почти всеки, който за пръв път се сблъсква с Python (или [CoffeeScript](https://en.wikipedia.org/wiki/CoffeeScript), или [YAML](https://en.wikipedia.org/wiki/YAML)). Въпреки, че в началото е малко странно как структурата на кода се определя с идентация/отместване, тази особеност на езика всъщност много допринася за ясния и четим код. Отговорът е: "Скобичките са в другите езици за програмиране :о)"

##### **Предпочитат се думите пред странните символи**

Да - в Python можете да напишете a & b и a | b (понякога с неочаквани резултати), но когато става дума за **логическите оператори** "и" и "или", винаги се предпочита a and b и a or b.

##### **други ... няма :о)**

добре де, излъгахме малко, но ще ги видим в течение на курса.

## [**Задача: 1. Инсталирайте Python на Вашия компютър**](http://python3.softuni.bg/student/lecture/assignment/56892782131b16099c63096c/?back=%2Fstudent%2Flecture%2F567ffa1c131b1642faa73795%2F)

### Всичко ли е наред с инсталацията?

За да проверите дали всичко е наред с инсталацията, напишете:

python3

Ако Python3 е инсталиран коректно, би трябвало да сте в интерактивната конзола на Python, която изглежда така:

Python 3.5.1 (default, ..............)

[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>

За да излезете от интерактивната конзола, натиснете Ctrl+D или напишете exit()

## Стартиране на Python в интерактивен режим (REPL)

За да стартирате Python в **интерактивен режим (REPL)** е достатъчно да го стартирате без параметри.

Под Linux & MacOS в терминала напишете:

python3

Под Windows е най-лесно да стартирате специализираната Python конзола и да напишете:

python

## Първи пример с Python 3

import turtle

turtle.speed('fastest')

colors = ['red', 'green', 'blue', 'purple']

for a in range(4):

turtle.color(colors[a])

x = 1

for b in range(1, 100):

turtle.forward(10)

turtle.right(100 - x)

x += 1

## Пробвайте сами

Полезни операции с костенурката:

* turtle.forward(length) - чертае права линия напред с дължина length като премества и костенурката
* turtle.backward(length) - чертае права линия назад с дължина length като премества и костенурката
* turtle.left(degrees) - завърта костенурката наляво указания брой градуси.
* turtle.right(degrees) - завърта костенурката надясно указания брой градуси.
* turtle.color('red') - сменя цвета на чертане на костенурката; всички рисувания след това ще използват този цвят; За цветове можете да използвате 'black', 'red', 'green', 'blue', 'orange', 'yellow', 'purple', и всички именовани цветове от HTML/CSS; също така можете да използвате и HTML/CSS цветове, напр: '#BA3290'.
* turtle.speed('fastest') - задава скоростта, с която костенурката рисува; можете да използвате 'fastest', ''fast'','normal', 'slow' и 'slowest'.

Пример за turtle в **интерактивната конзола (REPL)**:

$ python3

….

>>> from turtle import \*

>>> forward(120)

>>> left(140)

>>> forward(100)

>>> color(‘red’)

>>> left(140)

>>> forward(100)

За по-подробна информация: [Python 3 official documentation - turtle — Turtle graphics](https://docs.python.org/3/library/turtle.html)

## Променливи и проста аритметика в REPL на Python

DEMO

## Инструменти за писане на Python приложения

За да пишем приложения с Python е достатъчно да имаме произволен текстов редактор, който може да записва текстови файлове. За улеснение, обаче, е доста по-добре да използваме специализирани инструменти, които да ни подпомагат в тази дейност.

Предлагаме Ви кратко описание на вариантите, които ще използваме по време на курса - текстов редактор за писане на код, и PyCharm - интегрирана среда за разработка (IDE) за Python.

### Използване на текстов редактор за писане на код

За да пишете удобно и лесно на Python, **без да използвате специализирана среда за разработка (IDE)**, можете да използвате някои от популярните текстови редактори за код, които поддържат syntax highlight на Python.

* **SublimeText** - <http://www.sublimetext.com/3> - един от най-мощните текстови редактори; работи под Windows, Linux & MacOS.
* **GitHub Atom** - <https://atom.io/> - работи под Windows, Linux & MacOS
* **Microsoft Visual Studio Code** - <https://code.visualstudio.com/> - работи под Windows, Linux & MacOS
* **Komodo Edit** - <http://komodoide.com/komodo-edit/> - работи под Windows, Linux & MacOS
* **vim** - <http://www.fullstackpython.com/vim.html> - работи под Linux, MacOS & Windows
* **Notepad++** - <https://notepad-plus-plus.org/> - работи само под Windows

За да стартирате написаната програма:

Под Linux & MacOS :

python3 your\_program.py

Под Windows е най-лесно да стартирате специализираната Python конзола и да напишете:

python your\_program.py

## [**Задача: 2. Инсталирайте PyCharm**](http://python3.softuni.bg/student/lecture/assignment/568ac2b96e8efb7988e8de58/?back=%2Fstudent%2Flecture%2F567ffa1c131b1642faa73795%2F)

### Кратко обяснение как се работи с PyCharm

* Проект
* Стартиране на файл

## Други среди за разработка

Освен описаните по-горе, има и други среди за разработка за Python, които можете да изпробвате, ако имате желание:

* [Python Tools for Visual Studio](https://www.visualstudio.com/en-us/features/python-vs.aspx)
* [PyDev for Eclipse](http://www.pydev.org/)
* [Wing IDE](https://wingware.com/)
* IDLE - част от инсталацията на Python.

## Основи на Python

### Типове данни

* Основни типове данни int, float, str, bool, set, list, dict, None
  + int - произволно големи цели числа
  + float - числа с плаваща запетая
  + bool - булеви стойности - True и False
  + str - символни низове
  + bytes - последователност от байтове
  + tuple - наредена последователност от стойности, които могат да бъдат от един или различни типове;
  + list - списък (познат в други езици като "масив") от стойности, които могат да бъдат от един или различни типове;
  + dict - речник (познат в други езици като "dictionary", "Hashtable", "асоциативен масив" и т.н.), където за всеки уникален ключ има съответстваща стойност; ключовете и стойностите могат да бъдат от едни или различни типове; има ограничения за това какво може да се използва като ключ, което ще разясним в следващите лекции;
  + set - множество от неповтарящи се стойности от един или различни типове; има ограничения за това какво може да бъде елемент на множествата, което ще разясним в следващите лекции;
  + None - нулев обект (познат в други езици като "null", "nil" и т.н.)
* Всичко в Python е обект

### Променливи

demo

* a,b = b,a
* a, b, c = b, a, c

### Аритметика

demo

* деление на цели цисла и разликата между / и //
* деление на числа при които поне едното е float
* големи числа
* степенуване
* modulus % - остатък от целочислено делене
* divmod
* round, math.trunc, math.ceil, math.floor
* операции с несъвместими типове

### Условни оператори

demo

* if
* elif
* else
* Няма switch оператор в Python

### Логически оператори

demo

* ==, <, <=, >, >=, !=
* and, or

### Преобразуване на типове

demo

* str(),int(),float(),bool()
* Показваме какво се случва при несъвместими типове

### Коментари в кода

* Едноредов
* Многоредов

## [**Задача: 3. Нарисувайте квадрат**](http://python3.softuni.bg/student/lecture/assignment/567ffa4b131b1642fba73795/?back=%2Fstudent%2Flecture%2F567ffa1c131b1642faa73795%2F)

## Прост вход и изход

demo

* print(a)
* print(a,b)
* input

## Цикли

demo

* for
* while
* for i in range(0,10,3)
* break,continue

## Задача: Рисуване в цикъл по зададени ъгъл и дължина на линията

## Import

* Просто обяснение
* import X ; from X import Y ; from X import \*

## [**Задача: 5. Модифицирайте кода, така че да спре след определен брой итерации**](http://python3.softuni.bg/student/lecture/assignment/5684229e131b1653a054424c/?back=%2Fstudent%2Flecture%2F567ffa1c131b1642faa73795%2F)

## [**Задача: 6. Модифицирайте кода, така че да рисува интересни неща**](http://python3.softuni.bg/student/lecture/assignment/568527b5131b1653a054424d/?back=%2Fstudent%2Flecture%2F567ffa1c131b1642faa73795%2F)

## [**Задача: 7. Нарисувайте дъска за шах**](http://python3.softuni.bg/student/lecture/assignment/568534e9131b1653a054424e/?back=%2Fstudent%2Flecture%2F567ffa1c131b1642faa73795%2F)

## Полезни връзки

#### Материали и курсове за Python

Това са само много малка част от достъпните в Интернет курсове, книги и материали, от които можете да допълните знанията си за Python:

* [Dive into Python 3](http://www.diveintopython3.net/) - един от най-добрите ръководства, от които можете да научите Python;
* [Курс за Python от CodeAcademy](https://www.codecademy.com/learn/python)
* [Google's Python Class](https://developers.google.com/edu/python/?hl=en)
* [A Byte of Python](http://www.swaroopch.com/notes/python/)
* [Introduction to Programming with Python от OpenTechSchool](http://opentechschool.github.io/python-beginners/en/index.html)
* [Python Practice Book](http://anandology.com/python-practice-book/index.html)
* [Cheatsheet за разликите между Python 2 и Python 3](http://ptgmedia.pearsoncmg.com/imprint_downloads/informit/promotions/python/python2python3.pdf)

#### Online интерпретатори за Python

* [Online изпълнение на Python код на официалния сайт на езика](https://www.python.org/shell/)
* [Online изпълнение на Python код от repl.it, който поддържа и много други езици](https://repl.it/languages/python3)
* [Online изпълнение на Python код със Skulpt - Python интерпретатор в browser-а.](http://www.skulpt.org/)

# 2. Основни структури от данни - str, list, tuple, set, dict; Character encoding

## str

### Едноредов стринг

string, или както се казва в Python - str, е последователност от символи. Дефинира се, подобно на други езици, като стойността се огражда в кавички, като Python не прави разлика между единични ' или двойни " кавички.

sentence1 = 'стойността се огражда в кавички'

sentence2 = "Python не прави разлика между единични или двойни кавички"

Ако на два отделни реда имате два стринга **без нищо освен whitespace** между тях, Python ще ги обедини в един. Това е удобен начин да пренесете дълги стрингове на няколко реда.

>>> sentence\_long = 'стойността се огражда в кавички'

'Python не прави разлика между единични или двойни кавички'

print(sentence\_long)

'стойността се огражда в кавичкиPython не прави разлика между единични или двойни кавички'

Както виждате, при това положение Python няма да прибави интервал между двата стринга - ... кавичкиPython ....

Ако се налага да имате същата кавичка, с която сте дефинирали стринга, като символ в самия string, трябва да я "escape"-нете с наклонена черта.

sentence2 = "Python не прави разлика между единични ' или двойни \" кавички"

sentence2 = 'Python не прави разлика между единични \' или двойни " кавички'

### Многоредов стринг

За да дефинирате многоредов стринг, използвайте записа с 3 кавички - без значение единични или двойни.

sentence3 = """

За да дефинирате

многоредов стринг,

използвайте записа с 3 кавички

- без значение единични или двойни.

"""

При този запис Python ще запази новите редове.

Често този запис се използва и за коментиране на код на няколко реда едновременно, тъй като в Python липсва синтаксис за многоредови коментари.

### Методи на обекта str

По-долу са дадени най-често използваните методи на str обекта.

Пълно описнание можете да намерите в [документацията на Python](https://docs.python.org/3/library/stdtypes.html#string-methods), както и в autocomplete в PyCharm.

#### Търсене

* index(sub[, start[, end]]) и rindex() - връща индекса на първото срещане на под-стринга в стринга, но ако не бъде намерено съвпадение вдига exception
* find(sub[, start[, end]]) и rfind() - връща индекса на първото срещане на под-стринга в стринга, но ако не бъде намерено съвпадение връща "-1"
* startswith(prefix[, start[, end]]) - връща булев резултат (True/False), ако стрингът започва с подадения под-стринг
* endswith(suffix[, start[, end]]) - връща булев резултат (True/False), ако стрингът свършва с подадения под-стринг

>>> sentence4 = 'най-често използваните методи на str обекта.'

>>> sentence4.find('често')

4

>>> sentence4.find('между')

-1

>>> sentence4.index('често')

4

>>> sentence4.index('между')

ValueError Traceback (most recent call last)

....

----> 1 sentence4.index('между')

ValueError: substring not found

>>> sentence4.startswith('най-')

True

>>> sentence4.startswith('често')

False

>>> sentence4.endswith('обекта.') # с точка

True

>>> sentence4.endswith('обекта') # без точка

False

#### Заместване, изчистване, разделяне и промяна

Можете да съедините съдържанието на два или повече str с оператор +.

Методите, които 'променят' стринга, всъщност връщат **нов обект str**!

* replace(old, new[, count])
* strip([chars]), lstrip([chars]) и rstrip([chars]) - изчиства всички срещания в начaлото и в края на стринга на някои от символите в параметъра chars; ако не се подаде chars, се изчистват 'whitespace' символите - интервал, таб, нови редове и други;
* upper(), lower()
* join(arg) - връща нов string при който всеки елемент от arg е комбиниран със самия string. Този метод ще разгледаме по-подробно малко по-късно.

>>> sentence5 = "Методите на str, които 'променят' стринга, всъщност връщат нов обект str"

>>> sentence5.replace('str', 'стринг') # замества всички срещания

"Методите на стринг, които 'променят' стринга, всъщност връщат нов обект стринг"

>>> sentence6 = ' \t интервал, таб, нови редове \n\n\t '

>>> sentence6.strip()

'интервал, таб, нови редове'

>>> sentence6.lstrip()

'интервал, таб, нови редове \n\n\t '

>>> sentence6.rstrip()

' \t интервал, таб, нови редове'

>>> sep = ' , '

>>> sep.join(["A" , "B" , "C"])

'A , B , C '

#### slicing и разделяне

* s[idx] с положителна стойност на idx - връща idx-тия символ от началото на стринга (отпред назад); индексите започват от 0
* s[-idx] с отрицателна стойност на idx - връща idx-тия символ от края на стринга (отзад напред); -1 - последния символ,-2 - предпоследния символ
* s[from:]
* s[from:to]
* s[:to]
* s[:to:step]

>>> sentence7 = "Let's look at what's happening here. Just as before, we're specifying that we want to start at position 4 (zero-based) in the string"

>>> sentence7[11]

'a'

>>> sentence7[-1]

'g'

>>> sentence7[-2]

'n'

>>> sentence7[11:]

"at what's happening here. Just as before, we're specifying that we want to start at position 4 (zero-based) in the string"

>>> sentence7[11:35]

"at what's happening here"

>>> sentence7[:35]

"Let's look at what's happening here"

>>> sentence7[11:55:2]

'a htshpeighr.Js sbfr,w'

* split(sep=None, maxsplit=-1) - разделя стринга на указаните разделители, и връща списък с разделените подстрингове по подразбиране sep е whitespace (интервали, таб, нови редове и т.н.)
* splitlines() - разделя стринга на линии

>>> sentence8 = 'Python enables programs to be written compactly and readably.'

>>> sentence8.split()

['Python', 'enables', 'programs', 'to', 'be', 'written', 'compactly', 'and', 'readably.']

>>> sentence8.split('o')

'Pyth', 'n enables pr', 'grams t', ' be written c', 'mpactly and readably.']

>>> sentence9 = 'Beautiful is better than ugly. \r\n Beautiful is better than ugly. \n Readability counts. '

>>> sentence9.splitlines()

['Beautiful is better than ugly. ',

' Beautiful is better than ugly. ',

' Readability counts. ']

#### форматиране

Форматирането на стрингове е удобен начин за вмъкване на стойности в стрингов шаблон.

**!!! ВНИМАНИЕ !!!** Никога, ама **НИКОГА** не използвайте само стрингово форматиране с данни от външни източници, за да "сглобите" SQL заявки към бази данни, за да създадете HTML документи, да извиквате външни команди и т.н. Този тип грешки се наричат с общото наименование "Injection", и е най-често срещаната уязвимост в днешните софтуерни системи. Същото, разбира се, важи и за "сглобяване на стрингове с плюсчета".

След това стряскащо предупреждение, да видим малко прости примери:

"Today is {} °C outside".format(-8)

Можем да използваме и keyword argumens (за които ще говорим по-подробно в следващата лекция):

"Today is {temp} °C outside".format(temp=-8)

str.format() е доста мощен метод, но в голяма част от случаите простото форматиране, показано по-горе, върши работа. По-любопитните могат да погледнат [Format String Syntax](https://docs.python.org/3.5/library/string.html#format-string-syntax) и [Format Specification Mini-Language](https://docs.python.org/3.4/library/string.html#formatspec).

**Старият начин** за стрингово форматиране в Python използва операторът % и много прилича на printf() от C:

"Today is %s °C outside" % -8

Този начин на стрингово форматиране е доста често използван, особено в Python2, както и в по-стар код, но има някои недостатъци. Ако Ви е интересно, можете да погледнете [printf-style String Formatting](https://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting)

Полезни линкове:

* [PyFormat Using % and .format() for great good!](https://pyformat.info/)

#### isxxxx

Полезни за бърза проверка на съдържанието на стринга:

* .isdigit()
* .isalnum(), .isalpha()
* .islower(), .isupper(),

>>> '56'.isdigit()

True

>>> 'Beautiful is better than ugly'.islower()

False

>>> 'readability counts'.islower()

True

### По-подробно за encoding и Unicode

#### Какво означава "encoding"

"encoding" - правила за интерпретиране на последователност от байтове в определен контекст.

"Encoding" е общ термин, който не важи само за текстова, но и за много други видове информация - аудио, видео, структури от данни и т.н.

#### ASCII

В зората на компютрите, където най-вече развитието е в САЩ, първо възниква необходимостта от обработка на текстова информация. ASCII е прост encoding, който съпоставя числата от 0 до 127 (7 бита) към символи. Оригиналният ASCII включва

* от 0 до 31 и 127 - контролни символи, основно използвани в ранните терминали, както и символите за нов ред (#13 или 0x0D) и line feed (#10 или 0x0A).
* от 32 до 47, от 58 до 64, от 91 до 96 - пунктоационни символи - !, ?, $ , @, [, } и други;
* от 48 до 57 - цифрите от 0 до 9
* от 65 до 122 - буквите от английската азбука

В един момент изключително изненадващо се оказало, че на света не е само английската азбука, и се въвежда "Extended ASCII", който вече е 8-битов, и при който числата от 0 до 127 съвпадат с интерпретацията на оригиналния ASCII, а от 128 до 255 са поместени основно символи от западно-европейските езици - ü, ä, ê, ç, é и т.н., както и графични символи за рамки и запълване на местата на символите - ╤, ╟, ╔ и други.

Останалите хора по света - например България и Русия, където не се използват тези символи, предефинират Extended ASCII и създават множество свои encodings, при които символите от 128 до 255 представляват буквите от азбуката на кирилица.

Особено с развитието на Интернет в средата на 90-те години става ясно, че е невъможно нормална работа с интернационални документи, докато за представянето на символи се използва само 1 байт.

#### Какво е UCS & Unicode?

**UCS** и **Unicode** са **ISO/IEC** стандарт, описващ начина на електронно запазване на символи, знаци, цисла, идеограми, логограми. Това е основният стандарт който е приет от всички по света - включително Кирай и Северна Корея За разлика от **ASCII** където може да имаме само 256 "символа", Unicode може да съдържа 1,114,112 уникални символа. В момента Unicode съдържа "само" ~120,737 символа. Това включва всяка една известна писменост в човешката история, включително и всички китайски йероглифи (около 40000). Допълнително Unicode има огромен набор от научни символи (∫,∊), общоизвестни символи (卍,☭,☮,☪), емотиконки (☺,❤,☃) и какво ли още не - символ изобразяващ купчина фекалии.

При ASCII беше почти невъзможно в един документ да имаш текст на няколко езика, но в Unicode това не е никакъв проблем.

Най-важното нещо, което трябва да разберете за Unicode е, че всички съвременни компютърни системи използват тази кодова таблица. Използвайте я и Вие, за да сте съвместими с Живота, Вселената и всичко останало.

#### UTF-8

Unicode е универсален, но заема много място. Всеки символ се описва с приблизително 4 байта (32 бита), което го прави неефективен откъм запазване на информацията и прехвърлянето й по мрежа.

**UTF-8** е енкодинг, при който символите от Unicode се представят (кодират) така, че символите от оригиналния ASCII (включващи английската азбука и основните символи и препинателни знаци) заемат 1 байт, а тези от най-често използваните езици заемат 2 байта, останалите заемат 3, 4 или повече байта.

Тъй като стойностите за символите от оригиналния ASCII съвпадат с тези в UTF-8 ( напр. стойност 65 представя символа А, стойност 122 представя символа z) , всеки файл, записан в оригиналния ASCII encoding е и валиден UTF-8.

Тъй като символите от масово използваните езици заемат само 1 или 2 байта, това позволява ефективно запазване и пренос на текстове.

Работата с UTF-8 е трудна (поради променливия брой байтове, необходими за представянето на различните символи), и за това много програми и езици вътрешно конвертират UTF-8 в Unicode и работят директно с Unicode. Предимството на този подход е, че прави изпълнението на програмата много по-ефективно, защото символите са с един и същ размер, но недостатъкът е, че програмата има нужда от повече RAM.

Най-важното нещо което трябва да разберете за UTF-8 е, че един символ може да заема с 1 байт, а друг символ може да заема 3 . Кирилицата заема 2 байта за всеки символ, така че думата **Чушкопек** ще има нужда от 16 байта в UTF-8.

#### UTF-16 и други често използвани encodings

Освен UTF-8, има и други енкодинги като UTF-7, UTF-16, UTF-32, но най-масовият е UTF-8. Той е много ефективен когато текста се състои основно от английски или западноевропейски символи, но ако по-голямата част от текста е съставен от йероглифи или други символи, UTF-16 е по-подходящ.

Също така UTF-16 е по-ефективен за представянето на символите в паметта.

UTF-7 трябва да бъде забранен с международна конвенция под заплаха от нападение с ядрено оръжие, ако някой го използва. Не го използайте никога, освен ако не Ви е опрян пистолет в главата.

#### Полезни линкове

* [Много полезно видео как работи UTF-8](https://www.youtube.com/watch?v=MijmeoH9LT4)
* [Wikipedia - UTF-8](https://en.wikipedia.org/wiki/UTF-8)
* [Wikipedia: Universal Coded Character Set (UCS)](https://en.wikipedia.org/wiki/Universal_Coded_Character_Set)

### А в Python?

Python 3 поддържа пълния набор от UCS с размер на символите до 4 байта, като интерпретаторът преценява как да представи string-а, в зависимост от представянето на символа с най-голям ред. Например, ако string-ът съдържа само ASCII символи (от латинската азбука, цифри и основните препинателни знаци), Python ще използва само по 1 байт за съхранение на стринга; ако string-ът съдържа и символи на кирилица - целият string ще използва по 2 байта за всеки символ, и т.н. Тази хитрост позволява оптимално използване на паметта на компютъра.

##### **а UTF-8?**

За целта първо трябва да обясним за bytes :o)

## bytes

**bytes** е структура от данни, представляваща последователност от байтове. **Не символи, а байтове**.

Използва се при вход/изход от външни източници на информация, както ще покажем в следващите лекции.

info\_read\_from\_a\_file = b'\xd0\x90! \xd0\x9a\xd0\xb8\xd1\x80\xd0\xb8\xd0\xbb\xd0\xb8\xd1\x86\xd0\xb0, \xd0\xba\xd0\xbe\xd0\xb4\xd0\xb8\xd1\x80\xd0\xb0\xd0\xbd\xd0\xb0 \xd0\xb2 UTF-8 :\xd0\xbe)'

Ако стойността на байт-а съвпада със стойност от ASCII номера на символ, който може да се изпечата на конзолата, Python ще го покаже като нормална латинска буква; в противен случай записът е \xHH , къдeto HH е шестнайсетично число от 00 до FF. Както виждате от примера по-горе, в bytes обекта се чете 'UTF-8', както и интервалите, удивителната, ':' и ')'.

#### и к'во го правим това?

'това' е обикновено информацията, която ще прочетете, ако отворите нормален текстов файл, кодиран в UTF-8. За да можете да направите нещо смислено с прочетеното, трябва да 'декодирате' информацията съгласно encoding, с който е била записана.

В Python 3 това става така:

print(info\_read\_from\_a\_file.decode('utf-8'))

И понеже 'UTF-8' е най-популярният character encoding, това прави абсолютно същото:

print(info\_read\_from\_a\_file.decode())

Методът decode() преобразува байтове към str (или текстова инфорамция);

#### а наопаки?

"опаки" в случая е кодирането (encoding) на текстовата информация във вид, в който ще може да бъде записана във файл, и след това прочетена от някой друг ( дано :о) ).

Изненадващо, в случая използваме методът encode() на str:

info\_to\_be\_written\_out = 'Да видим това дали ще можем да го прочетем после?'.encode('utf-8')

И, както и при decode(), ако не се подаде изрично име на encoding, се приема UTF-8:

info\_to\_be\_written\_out = 'Да видим това дали ще можем да го прочетем после?'.encode()

В следващите лекции ще пострадаме по-сериозно от character encoding.

### [Задача: 1. Ограничете дължината на текст](http://python3.softuni.bg/student/lecture/assignment/5686c81b131b1653a054424f/?back=%2Fstudent%2Flecture%2F568014cc131b1642fba73796%2F)

### [Задача: 2. Вземете текста след определен символ](http://python3.softuni.bg/student/lecture/assignment/5686cbbd131b1653a0544250/?back=%2Fstudent%2Flecture%2F568014cc131b1642fba73796%2F)

### [Задача: 3. Инициали](http://python3.softuni.bg/student/lecture/assignment/568913de131b1603e789a80b/?back=%2Fstudent%2Flecture%2F568014cc131b1642fba73796%2F)

## list

list е подредена структура, като елементите могат да бъдат от различни типове, и да се повтарят.

values = [1, 5.6, 'dve']

DEMO

##### **Стандартни операции**

... познати от почти всеки език за програмиране:

* индексиран достъп до елементите - values[2] # == 'dve', като индексите започват от 0
* присвояване на нова стойност на елемент по индекс - values[3] = 'друга стойност'
* добавяне на елемент в края - values.append(784)
* вмъкване на елемент - values.insert(0, 123), ще вмъкне стойността 123 на първо място в списъка
* премахване на елемент - values.remove('dve') - премахва първото срещане на подадения елемент от списъка (не по индекс)
* премахване на елемент по индекс - values.pop(2) - премахва елемента на подадения индекс, и връща стойността като резултат
  + можете да използвате и del values[2] , който обаче не връща премахнатата стойност
* sort() - сортира елементите в list-а
* reverse() - обръща поредността на елементите в list-а
* търсене на елемент - values.index(5.6)
* изчистване - values.clear()

##### **Допълнителни операции с list, налични в Python**

* преброяване на срещанията на елемент в list-а - values.count('dve')
* проверка дали елемент се среща в списъка:
  + 'dve' in values
  + 'dve' not in values
* събиране на списъци - values + [7, 11, 13, 17]
* повтаряне на списък - [7, 11, 13, 17] \* 2
* **slicing**: при values = [1, 2, 3, 4, 5, 6, 7, 8]
  + values[1:]
  + values[2:5] - елементите от индекс 2 (включително) до индекс 5 (изключващо)
  + values[:7] - елементите до индекс 7 (изключващо)
  + values[::2] – всеки втори елемент
  + values[-1] - последния елемент
  + values[-6: -2] - част от списъка, като индексирането е отзад напред
  + values[-6: -2: 2] - всеки втори част от списъка, като индексирането е отзад напред
  + . . . :о)

##### **Итериране на list**

for value in values:

print(value)

Ако ни е необходим и индекса на съответната стойност, можем да използваме enumerate()

for idx, value in enumerate(values):

print(idx, ' - ', value)

### [Задача: 4. Средна цена, както се прави в белите държави](http://python3.softuni.bg/student/lecture/assignment/5689178b131b16073d076470/?back=%2Fstudent%2Flecture%2F568014cc131b1642fba73796%2F)

## tuple

tuple може да съдържа стойности от различен тип (също както list), но веднъж създаден, не може да бъде променян (immutable). Това е и причината tuple да изисква много по-малко ресурси от list.

values = (2, 3, 'a')

DEMO

tuple поддържа само методите, които не променят съдържанието - count() и index(). Също така tuple поддържа и **slicing**, както е показахме преди малко за list.

Python позволява т.нар. **expansion** на стойности от тип tuple:

a, b = (2, 3)

а по-новите версии на езика поддържат и **wildcard tuple expansion**:

a, b, \*rest = (4, 5, 6, 7, 8)

# a == 4

# b == 5

# rest == [6, 7, 8]

##### **Итериране на tuple**

for value in (1, 2, 4, 8, 16):

print(value)

Ако ни е необходим и индекса на съответната стойност, можем да използваме enumerate()

for idx, value in enumerate((1, 2, 4, 8, 16)):

print(idx, ' - ', value)

## set

Множеството (set) е съвкупност от уникални елементи. При него **няма дефинирана подредба**, а основната операция е**принадлежност на елемент към множеството**.

x = {1, 2, 3}

y = {3, 4, 5}

z = set(9, 10)

DEMO

Проверка за принадлежност:

>>> 5 in x

False

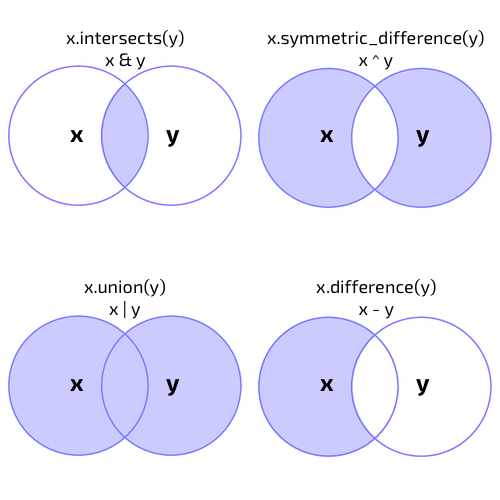
>>> 5 not in x

True

>>> 5 in y

True

##### **Основни операции с множества**



В Python е за предпочитане да използвате методите, тъй като са по-ясно четими за не-математици.

##### **Итериране на set**

for value in x:

print(value)

Ако ни е необходим и индекса на съответната стойност, можем да използваме enumerate()

for idx, value in enumerate(x):

print(idx, ' - ', value)

**!! ВАЖНО !!** Имайте предвид, че за set нямаме гарантирана последователност, т.е. итерацията на елементите може да се случи в произволен ред.

### [Задача: 5. Общи интереси](http://python3.softuni.bg/student/lecture/assignment/56891a82131b16073d076471/?back=%2Fstudent%2Flecture%2F568014cc131b1642fba73796%2F)

## dict

dict() (речник) представлява вградена структура от данни, която ни позволява да правим асоциация между ключ и стойност. В други езици е познат като Hashtable, асоциативен масив, hybrid dictionary, hash и т.н. Например, искаме да запишем каква е температурата в различни градове, така че в последствие да можем да работим лесно с тази информация. В този случай името на града ще е ключа, а температурата ще е стойността на dict:

София: -14

В един речник може да имаме множество ключове/стойности:

Новосибирск: -31

Таити: 30

Най-лесният начин да създадем нов речник в Python е:

weather={

'София': -14,

'Новосибирск': -31,

'Таити': 30

}

Ако искаме да вземем температурата за определен град, използваме ключа (името на града):

temp = weather['София']

Ако искаме да променим стойност, отново използваме ключа:

weather['София'] = -2

Накратко, ако искаме да правим нещо с дадена стойност, трябва да се обърнем към нея използвайки ключа.

Какво ще стане, ако се опитаме да присвоим стойност на несъществуващ ключ:

weather['Ахваз'] = 46

Ами нищо, така добавяме нов запис в речника. Ако ключа съществува новата стойност ще се асоциира с него, ако не съществува ще се създаде нов.

Има няколко особености, с които трябва да се съобразявате, когато работите с речици:

* Ключа трябва да бъде hashable елемент. Ако не разбирате какво точно означава hashable, не се притеснявайте - текст, цели числа, булеви стойност, tuples, sets са hashable. **Никога не си позволявайте са използвате числа с плаваща запетая за ключове** - може да имате доста неочаквани реултати, и почти невъзможни за откриване грешки;
* Стойностите може да са всеки един възможен тип, който Python поддържа;
* Не може да има два или повече еднакви ключа; идеята на ключа е да бъде уникален в целия речник.
* Може да имате повтарящи се стойности;
* Търсенето по ключ е изключително бърза операция със сложност O(1). Ако не знаете какво е O(1), приемете, че Python може да работи ефективно с речници от порядъка на стотици хиляди ключове. Все пак за да сме максимално точни, понякога сложността за серия ключове може да е O(n);
* Не може директно да търсите по стойности в речниците - необходим е ключ, за да вземете която и да е стойност;
* Речниците нямат подредба -Python не гарантира, че елементите ще се итерират в същия ред, в който сте ги въвели. Ако имате нужда от подреден речник, Python предлага такава структура, която се нарича OrderedDict

Ето някой от най-често използваните методи за работа с речници

my\_dict = dict() # създава нов празен речник

my\_dict = {} # идентично с горното. Това е приетият начин в света на Python да се създават нови речници

len(my\_dict) # връща броя на елементите (ключовете) в речника

del my\_dict['key'] # премахва ключа и неговата стойност от речника

'my\_key' in my\_dict # връща True ако 'my\_key' съществува в речника, независимо от стойността, False ако го няма

'my\_key' not in my\_dict # връща False ако 'my\_key' съществува в речника, независимо от стойността, True ако го няма

my\_dict.clear(my\_dict) # премахва всички ключове и стойности от речника

my\_dict.get('my\_key') # връща стойността за 'my\_key'. Ако 'my\_key' не съществува се връща None

my\_dict.get('my\_key', 'default\_value') # връща стойността за 'my\_key'. Ако 'my\_key' не съществува се връща стойността подадена като втори параметър, в случая текста 'default\_value'

my\_dict.keys() # връща 'списък' с всички ключове в речника. Стойностите не се връщата

my\_dict.values() # връща 'списък' с всички стойности в речника, ключовете не се връщат

Речниците могат да бъдат итерирани:

weather = {

'София': -14,

'Новосибирск': -31,

'Таити': 30

}

for key in weather:

print(key) # при всяка итерация променливата key ще бъде един от ключовете в речника

print(weather[key]) # тъй като имаме ключа може да вземем и стойността асоциирана с него

for key, value in weather.items():

print(key)

print(value)

for value in weather.items():

print(value)

### [Задача: 6. Matchmaking](http://python3.softuni.bg/student/lecture/assignment/56890019131b1603e789a80a/?back=%2Fstudent%2Flecture%2F568014cc131b1642fba73796%2F)

### [Задача: 7. Matchmaking - v2](http://python3.softuni.bg/student/lecture/assignment/56892466131b16073d076472/?back=%2Fstudent%2Flecture%2F568014cc131b1642fba73796%2F)

# 3. Функции и scope

## Функции

### Дефиниране на функция

Функциите в Python се дефинират с ключовата дума def, след което следва името на функцията, опционални параметри и тялото на функцията.

Пример:

def print\_temperature(temp):

print(temp, 'C')

Важните неща за именоване на функциите са:

* имената на функциите може да съдържат само букви, цифри и \_ - правилата са същите, както и за имена на променливи.
* имената на функциите НЕ може да започват с цифра
* прието е имената на функциите да се пишат в snake\_case с малки букви

Опитвайте се винаги да пишете кодa си така, че функциите да зависят единствено и само от параметрите, които са подадени, и да не променят състояния извън тялото на функцията - т.е. без странични ефекти.

Ако дефинирате две или повече функции с **едно и също име в един scope**, активна ще остане само последната. Причините за това се крият в специфичния начин, по който работи Python.

### Връщане на стойност

Функциите в Python могат да връшат резултат, използвайки ключовата дума return.

def convert\_fahrenheit\_to\_celsius(deg\_f):

return (deg\_f - 32) / 1.8

Важните неща за връщането на резултат от функциите са:

* Връщането не е задължително. Може да имаме функция, която не връща нищо. В този случай ако се опитате да присвоите резултатът на променлива, резултатът ще бъде None
* Връщаната стойност може да бъде от всеки един възможен тип, поддържан от Python
* След ключовата дума return, изпълнението на функцията се прекратява.

### Параметри

Функциите в Python могат да имат произволен брой параметри, и те могат да се използват САМО в тялото на функцията.

def bigger(param1, param2):

if param1 > param2:

return param1

return param2

print(bigger(10,15)) # резутатът ще е 15

#### Параметри със стойност по подразбиране

В Python всеки един параметър може да има стойност по подразбиране. Тази стойност ще се използва, когато този параметър не е подаден при извикването на функцията.

def bigger(param1,param2=22):

if param1 > param2:

return param1

return param2

print(bigger(101)) # резултатът ще е 101, защото param2 ще е със стойност 22

#### Именовани параметри

Python позволява да подаваме параметрите по име - не само по позиция.

Например, имаме функция send\_mail(), която в други езици бихме извикали така:

send\_email(

"Лекция за функции - за курса по Python",

"Здрасти, готови ли сме с лекцията за функции, или още не сме я написали?",

"gatakka@example.com",

"boris.chervenkov@example.com",

"gatakka@example.com",

None

)

в примера по-горе имаме 3 последователни параметъра, които съдържат email адрес, и от пръв прочит въобще не е ясно кой параметър за какво служи. За да разберем, или трябва да си помогнем с IDE-то, или да четем документация, или в краен случай да ровим в кода на функцията (ако е наличен, разбира се); с две думи губим си времето.

Същата функция в Python може да се извика и така:

send\_email(

subject="Лекция за функции - за курса по Python",

body="Здрасти, готови ли сме с лекцията за функции, или още не сме я написали?",

from\_email="gatakka@example.com",

to\_email="boris.chervenkov@example.com",

cc\_email="gatakka@example.com",

attachments=None

)

Доста по-ясно, нали?

Използването на именовани параметри е изключително удобно, води до много лесно четим и "self-documenting" код. Съветваме Ви да ги използвате често - ще ни благодарите по-късно :о)

Друго предимство е, че ако в бъдеще добавим нов параметър към функцията, старият код ще работи безпроблемно, понеже параметрите ще се определят от името, а не от позицията.

Също така, чрез използването на именовани параметри премахваме нуждата от спазване на точно определена последователност при подаването на им.

def function(param1 , param2):

print(param1) # param1 е 33, нищо че е подаден като 2-ри параметър

print(param2) # param2 е 10, нищо че е подаден като 1-ви параметър

function(param2 = 10 , param1 = 33)

Именованите параметри могат да работят със стойности по подразбиране:

def function(param1 = 8 , param2 = 13):

print(param1) # param1 е 8, защото не е подаден при извикване на функцията

print(param2) # param2 е 10, защото е подаден при извикване на функцията

function(param2 = 10)

Може да комбинираме позиционни и именувани параметри:

def function(param1 = -1 , param2 = 13):

print(param1) # param1 е 10

print(param2) # param2 е 87

function(10,param2=87)

Когато описвате параметри със стойност по подразбиране, е задължително те да се намират **СЛЕД** параметрите без стойност по подразбиране

def function(param1 , param2 = 20 , param3 = True): # вярно

. . .

def function(param2 = 20 , param1 , param3 = True): # грешно, МНОГО грешно

. . .

#### Променлив брой аргументи args и \*kwargs

Функциите в Python могат да имат и променлив брой позиционни и именовани параметри.

За да укажем, че имаме променлив брой параметри, при дефинирането на функцията използваме \*args като име на параметър.

def sum\_numbers(\*args): # args ще бъде tuple, който ще съдържа стойностите на всички подадени позиционни параметри

total = 0

for n in args:

total += n

return total

# ... и да - знаем, че в Python има много по-кратък начин да свършим същата работа :о)

print(sum\_numbers())

print(sum\_numbers(4))

print(sum\_numbers(5, 1, 49, 26, 45, 34, 3, 81))

Важна в случая е \* пред името на променливата - args е конвенция, и може да бъде произволен идентификатор - например\*slonove. Ако нямате добра причина за друго име, използвайте \*args.

С именованите параметри нещата стоят по подобен начин:

def pretty\_print\_record(\*\*kwargs): # във функцията kwargs ще бъде обикновен dict

print("Record:")

for k, v in kwargs.items():

print("\t", k, "= ", v)

pretty\_print\_record(name="Mercury", distance\_au=0.387, diameter\_km=4878)

pretty\_print\_record(name="Venus", distance\_au=0.723, diameter\_km=12104)

pretty\_print\_record(name="Earth", distance\_au=1, diameter\_km=12742, average\_temp\_c=7.2, atmosphere=["nitrogen", "oxygen", "argon"])

pretty\_print\_record()

Важна в случая е \*\* пред името на променливата - kwargs е конвенция, и може да бъде произволен идентификатор - например\*\*krokodili. Ако нямате добра причина за друго име, използвайте \*\*kwargs.

Тъй като \*\*kwargs е обикновен dict, можем да подобрим малко функцията от по-горе:

def pretty\_print\_record(\*\*kwargs): # във функцията kwargs ще бъде обикновен dict

print(kwargs.pop('name', "Record"), ":")

for k, v in kwargs.items():

print("\t", k, "= ", v)

pretty\_print\_record(name="Venus", distance\_au=0.723, diameter\_km=12104)

#### annotations за параметрите и връщаните от функциите стойности

Python е език с динамично типизиране на променливите. Понякога, обаче, не е излишно, ако можем да укажем какви типове данни очакваме, и какъв тип ще бъде върнатият от функцията резултат:

def convert\_fahrenheit\_to\_celsius(deg\_f: float) -> float:

return (deg\_f - 32) / 1.8

Анотациите в Python дават възможност към параметри или резултат от функция да бъде приложена мета информация, като самият език **не използва по никакъв начин тези анотации** - те са предназначени изцяло за разработчиците.

Обикновено, като конвенция, в анотациите се описват типове - int, str, float, и всеки друг тип, поддържан от езика.

Въпреки, че в настоящия курс няма да ни се налага да ги използваме, е добре да знаете какво означават, ако ги видите в друг код.

#### Извикване на функции с args & \*kwargs

Понякога е полезно да извикате функция, като не знаете точния брой и вид параметри.

В Python можете да го направите по следния начин:

def format\_with\_indent(format\_string, \*args, indent: int=None, indent\_with: str=" ", \*\*kwargs):

if indent is not None:

indent\_str = indent\_with \* indent # will multiply the indent string

else:

indent\_str = ""

return indent\_str + format\_string.format(\*args, \*\*kwargs)

...

print(format\_with\_indent("Name: {}, Role: {role}", 'Boris', role='lecturer'))

print(format\_with\_indent("Name: {}, Role: {role}", 'Boris', role='lecturer', indent=4))

print(format\_with\_indent("Name: {}, Role: {role}", 'Boris', role='lecturer', indent=4, indent\_with='-'))

## Scope

Подобно на останалите езици, Python работи с концепцията за **Scope** (видимост). Най-простото обяснение за Scope на функциите е: Всичко което се случи във функция остава във функцията.

counter = 1

def do\_some\_work(counter):

counter+=10 # променливата counter във функцията няма нищо общо с променливата counter извън функцията.

do\_some\_work(counter)

print(counter) # резултатът ще е 1

Функциите нямат директен достъп до променливи извън тях, както и код извън функции нямат диерктен достъп до кода във функциите.

counter = 1

def do\_some\_work():

counter+=10

do\_some\_work()

# ще получим UnboundLocalError: local variable 'counter' referenced before assignment грешка, понеже Python не знае какво е counter в контекста на функцията.

Ако искаме да позволим на функция да има директен достъп до дадена променлива извън тялото на функцията трябва изрично да кажам, че тази променлива е **глобална**

counter = 1

def do\_some\_work():

global counter

counter+=10

do\_some\_work()

print(counter) # резултатът ще е 11 понеже Python може да намери променливата counter извън тялото на функцията.

**Важно** - Използването на global в който и да е език за програмиране е много ясен индикатор, че нещо не е наред с архитектурата и логиката на Вашия код. Опитвайте се да пишете кода така, че да не Ви се налага да използвате global. Разбира се, понякога е неизбежно, но се постарайте в рамките на разумното да не го използвате.

# 4. Exceptions, debugging, текстови файлове

## Exceptions

Exceptions (изключения) са популярен механизъм за обработка на грешки в модерните езици.

При възникване на неочаквана грешка - например при опит за достъп до несъществуващ ключ в dict, или отваряне на несъществуващ файл, практиката в Python е да се "хвърли exception" / "raise exception", при което изпълнението на кода бива незабавно прекратено, и изпълнението се прехвърля към най-близкия блок за обработка на изключения.

Пример:

people = [{'name': "Мария", 'gender': "female", }, {'name': "Калоян", 'gender': "male", }, ]

...

def print\_person(person: dict):

print("{} ({}) is interested in {}".format(

person['name'],

person['age'],

', '.join(person['interests'])

))

def print\_people(people: list):

for person in people:

print\_person(person)

Ако се опитаме да извикаме функцията:

print\_people(people)

Traceback (most recent call last):

File "exception\_not\_handled.py", line 18, in <module>

print\_people(people)

File "exception\_not\_handled.py", line 14, in print\_people

print\_person(person)

File "exception\_not\_handled.py", line 8, in print\_person

person['age'],

KeyError: 'age'

Грешката възниква във функцията print\_person, където не се обработват грешки. В този случай изпълнението на print\_person()се прекратява, и се предава на функцията, която я е извикала - в случая print\_people. В print\_people също няма код за обработка на exception, и изпълнението на тази функция също се прекратява. Този процес се изпълнява докато не бъде достигнат код за обработка на exception-а, или докато не се стигне до стартирания файл, при което цялата програма спира изпълнението си.

### Прихващане и обработка на exceptions

Прихващането на exceptions в Python става с помощта на try / except блок:

try:

print\_person({"name": "Калоян"})

except:

print("Непълни данни за отпечатване!")

По този начин except блока ще бъде изпълнен само ако в тялото (между try и except) възникне exception. В примера по-горе показаният except блок ще прихване **всички** exceptions, които биха възникнали.

По-добра практика, особено в големи проекти, е да прихващате само специфични exceptions, които можете да обработите, например:

try:

print\_person({"name": "Калоян"})

except KeyError as e:

print("Непълни данни за отпечатване: " + str(e))

В този случай при възникване на KeyError ще се изпълни Вашия код за обработка на exception, а при друг exception изпълнението на функцията ще бъде прекратено по описания по-горе процес.

### "Хвърляне" на exception

Вашият код също може да хвърля exception, с помощта на ключовата дума "raise"

def print\_person(person: dict):

if 'age' not in person:

raise ValueError('Ключът "age" е задължителен в параметъра "person"')

print("{} ({}) is interested in {}".format(

person['name'],

person['age'],

', '.join(person['interests'])

))

ValueError е стандартен exception от библиотеката на Python.

За повече информация:

* [Python tutorial: 8. Errors and Exceptions](https://docs.python.org/3.5/tutorial/errors.html)
* [Python docs: 5. Built-in Exceptions](https://docs.python.org/3/library/exceptions.html)

## Logging

Стандартният начин за лог-ване на съобщения, грешки или друга информация за работата на програмата Ви е с помощта на пакета logging.

Най-простият начин за използване е:

import logging

...

logging.debug("Contents of x: ".format(5))

logging.info("File saved successfully")

logging.warning("Missing required field 'age' - ignoring the person")

logging.error("Unable to load data file")

logging.critical("Unable to start the program! Aboring execution")

Всеки един от тези методи log-ва съобщение с едно от стандартните нива - logging.DEBUG, logging.INFO, logging.WARNING,logging.ERROR и logging.CRITICAL.

При този начин на използване - без конфигурация, ще се показват само log–овете, които са с ниво logging.WARNING или по-високо.

### Предимства пред използване на прост print()

* с помощта на проста конфигурация можете да пренасочите определени или всички log съобщения към конзола, файл, специализирани системи за събиране и анализ на log информация, или да спрете всички log съобщения, **без да променяте кода на програмата си**;
* при четене е много по-ясно, че искате да покажете log съобщение; ако за всичко използвате print() е по-трудно да различите кое е log съобщение, и кое извежда реален резултат от изпълнение на програмата.

За повече информация: [Basic Logging Tutorial](https://docs.python.org/3/howto/logging.html#logging-basic-tutorial)

### Конфигурация

Ако желаете да виждате всички logging съобщения, без да се занимавате с конфигурация, можете да използвате:

import logging

logger.setLevel(logging.DEBUG)

...

logging.debug("Some message")

Опростеният начин за конфигурация на logging модула е чрез използване на функцията basicConfig()

import logging

logging.basicConfig(filename='task2.log',level=logging.DEBUG)

...

logging.debug("Some other message") # това съобщение ще бъде записано във файла 'task2.log'

Това ще конфигурира logging модула да записва log съобщенията във файл, като включва всички съобщения от нивоlogging.DEBUG и по-високи.

Подробна информация за използването на logging можете да намерите в документацията на Python:

* [Advanced Logging Tutorial](https://docs.python.org/3/howto/logging.html#advanced-logging-tutorial)
* [logging.basicConfig](https://docs.python.org/3/library/logging.config.html#module-logging.config)
* [Logging configuration](https://docs.python.org/3/library/logging.config.html#module-logging.config)

## Debugging

### С "подръчни материали"

Най-простият начин за debugging е чрез използване на logging (или print()) на ключови места в програмата, за да проверите какви са моментните стойности на променливите, и дали въобще някаква част от кода се изпълнява. За предпочитане е да използвате logging.debug().

Това е вероятно най-старият метод за debugging, но и до днес остава един от важните инструменти за проследяване на грешки в кода, особено когато става дума за работа на кода на сървър, където не можете да използвате IDE с графичен debugger.

### С използване на IDE

DEMO

Други полезни инструменти, които могат да Ви помогнат:

* [pdb - Python's debugger](https://docs.python.org/3.5/library/pdb.html)
* [pudb](https://pypi.python.org/pypi/pudb)

## Работа с текстови файлове

Текстовите файлове са един от основните механизми за съхранение и обмен на информация.

Пример за изпечатване на съдържанието на текстов файл:

f = open('./directory/lecture.txt')

for line in f:

print(line)

f.close()

Винаги, когато отворите файл, трябва да го затворите с .close(), след като свършите да четете или пишете в него. В Python има по-кратък и по-безопасен начин да работите с файлове:

with open('./directory/lecture.txt') as f:

for line in f:

print(line)

Този код прави абсолютно същото, както предишния, но при него има гаранция, че файлът ще бъде затворен, след като се излезе от тялото на with, и е по-кратък и разбираем. Как и защо работи този код ще разберете в една от следващите лекции.

За да **презапишете съдържанието** на файл, като **елиминирате текущото му съдържание**, при отваряне трябва да добавите допълнителен параметър - w:

with open('./directory/numbers.txt', 'w') as f: # с 'w' съдържанието на файла ще бъде изтрито, и ще започнем в пишем на празен файл

for i in range(20):

f.write(str(i))

f.write("\n")

За да **допълните съдържание** във файл, като **запазите текущото му съдържание**, при отваряне трябва да добавите допълнителен параметър - а:

with open('./directory/numbers.txt', 'а') as f: # с 'а' съдържанието на файла се запази, и писането в този файл ще допълва в края на файла

for i in range(20):

f.write(str(i))

f.write("\n")

**!! ВАЖНО !!** .write() няма автоматично да добавя нови редове при писане във файла.

Други полезни методи:

* .readline() - прочита една линия от отворения файл, като връща str, който включва разделителя за нов ред
* .readlines() - прочита всички линии от файла и връща list от str с всички прочетени линии. **ако файлът е прекалено голям**, с използването на този метод е възможно да заемете прекалено много памет;
* .read() - без параметър прочита цялото съдържание на файла, и връща str. **ако файлът е прекалено голям**, с използването на този метод е възможно да заемете прекалено много памет;
* .read(max\_chars) - прочита най-много указания брой символи от файла, и връща str

### [Задача: 2. Намерете средната цена на продукт от текстови файл](http://python3.softuni.bg/student/lecture/assignment/5696b2ee6e8efb1652742bd7/?back=%2Fstudent%2Flecture%2F5680152a131b1642fba73798%2F)

### [Задача: 3. Намерете средната цена на продукт от текстови файл, групирана по критерии](http://python3.softuni.bg/student/lecture/assignment/5696b4346e8efb165374296d/?back=%2Fstudent%2Flecture%2F5680152a131b1642fba73798%2F)

### [Задача: 4. Променете цените в каталог](http://python3.softuni.bg/student/lecture/assignment/5696b6036e8efb165374296e/?back=%2Fstudent%2Flecture%2F5680152a131b1642fba73798%2F)

# 5. Пакети, стандартни формати, os & sys, време

## Модули и пакети

### Модули

Python файл се нарича 'module'. Дефинираните в него функции, класове и променливи можете да използвате с import.

Резултатът от следните примери е един и същ:

import utils\_module

utils\_module.print\_with\_indent("Дефинираните в него функции, класове и променливи можете да използвате с `import`.", indent=8)

from utils\_module import print\_with\_indent

print\_with\_indent("Дефинираните в него функции, класове и променливи можете да използвате с `import`.", indent=8)

from utils\_module import print\_with\_indent as some\_fancy\_name

some\_fancy\_name("Дефинираните в него функции, класове и променливи можете да използвате с `import`.", indent=8)

from utils\_module import \* # не е добра практика - не го използвайте

print\_with\_indent("Дефинираните в него функции, класове и променливи можете да използвате с `import`.", indent=8)

При import на модул, кодът в него се изпълнява.

### Пакети

Пакетите в Python са механизъм, с който можете да организирате модули при по-големи проекти.

Пакетите се организират в директории, подобно на други езици (напр. Java). За да бъде дадена директория пакет, в нея трябва да има файл, който се казва \_\_init\_\_.py

Импортирането на пакети става по същия механизъм, както и импортирането на модули.

## Инсталиране на пакети

pip install package\_name

# стар метод

easy\_install package\_name

## Стандартни пакети os & sys

### os

Пакетът os съдържа множество полезни функции, които са свързани с работата на програмата с операционната система - информация за файлове, директории, както и за взаимодействие с операционната система на по-ниско ниво.

import os

if os.access('/etc/passwd', os.W\_OK):

print("Не би трябвало да можете да пишете в този файл")

elif os.access('/etc/passwd', os.R\_OK):

print("Можете да прочетете списъка с потребителите.")

Полезни функции за курса:

* os.walk() - позволява обхождане на съдържанието на директория и поддиректории;
* os.access() - проверява дали имате достъп до файл или директория;

Пример за обхождане на директория:

import os

for dirpath, dirnames, filenames in os.walk('./'):

print("Директория '{}' съдържа:".format(dirpath))

print("Под-директории: \n\t{}".format("\n\t".join(dirnames)))

print("Файлове: \n\t{}".format("\n\t".join(filenames)))

print("\n-------------\n\n")

### os.path

os.path предоставя функции за лесна работа с файлови пътища и имена.

import os

print(os.path.join("C:\Users", "Guest", "Documents", "document.docx"))

"C:\Users\Guest\Documents\document.docx"

Полезни функции в os.path:

* os.path.join - слепя няколко стринг-а така, че да се получи валиден път до файл;
* os.path.exists - проверява дали подаденото име съществува - независимо дали е файл или директория;
* os.path.basename - връща името на файла (с разширението) по подаден пълен път до файл;
* os.path.dirname - връща името на директорията по подаден пълен път до файл;
* os.path.getsize - връща размера на файл в байтове по подадено име;
* os.path.isfile - проверява дали подаденият параметър е име на съществуващ файл;
* os.path.isdir - проверява дали подаденият параметър е име на съществуваща директория;
* os.path.splitext - връща tuple с 2 елемента - (име на файла, разширение на името на файла);

### Параметри от командния ред, и sys

Стандартният пакет sys съдържа редица системни функции и информация, като пълно описание можете да намерите в документацията.

За целите на курса ще разгледаме само приемането на параметри на програмата от командния ред. В специалната променливаsys.argv Python ще попълни параметрите, които са подадени от командния ред.

arguments.py:

import sys

print("Параметри на програмата:")

print(sys.argv)

for idx, a in enumerate(sys.argv):

print("Argument #{} - '{}'".format(idx, a))

При извикване ще получите:

python3 arguments.py

Параметри на програмата:

['arguments.py']

Argument #0 - 'arguments.py'

python3 arguments.py filename.txt

Параметри на програмата:

['arguments.py', 'filename.txt']

Argument #0 - 'arguments.py'

Argument #1 - 'filename.txt'

python3 arguments.py filename.txt 46

Параметри на програмата:

['arguments.py', 'filename.txt', '46']

Argument #0 - 'arguments.py'

Argument #1 - 'filename.txt'

Argument #2 - '46'

На индекс 0 в sys.argv винаги е попълнено името на програмата, която е стартирана - в примерите по-горе - "arguments.py".

### [Задача: 1. Търсене във файловата система](http://python3.softuni.bg/student/lecture/assignment/5696a9bb6e8efb1652742bc3/?back=%2Fstudent%2Flecture%2F56801568131b1642fba73799%2F)

## Време и дати

Времето е сравнително проста концепция - движи се в една и съща посока, при това равномерно - секунда след секунда.

import time

print(time.time())

Най-често използваният метод за отчитане на времето в POSIX операционните системи е т.нар. UNIX timestamp - число, указващо брой секунди от 01.01.1970 до момента. Въпреки, че този метод е широко използван, когато е необходимо да се работи с дати, време и календар, е задължително да се използват специализираните средства на платформата, вместо да се работи директно с timestamp.

В Python инструментите за работа с дати и часове се намират в пакета datetime

Обектът datetime от пакета datetime работи с информация за час и дата, с включено управление и на часови зони.

from datetime import datetime

print(datetime.now())

print(datetime(2016, 1, 19))

print(datetime(2016, 1, 19, 20, 21, 22, 222425))

print(datetime(year=2016, month=1, day=19, hour=20, minute=21, second=22, microsecond=222425))

За разпечатване/форматиране на datetime стойности можете да използвате няколко метода:

d = datetime(2016, 1, 19, 20, 21, 22, 222425)

print(str(d))

2016-01-19 20:21:22.222425

print(d.strftime('година: %Y, месец: %m, ден: %d, час: %H минута: %M секунда: %S'))

година: 2016, месец: 01, ден: 19, час: 20 минута: 21 секунда: 22

print(d.isoformat())

2016-01-19T20:21:22.222425

Пълно описание на форматиращите команди за .strftime() можете да намерите [в документацията на Python3](https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior) или на този адрес<http://strftime.org/>

Важно е да се знае, че много от форматиращите команди зависят от locale (текущите регионални настройки).

Конвертирането на стойност от str към datetime обект става със strptime().

string\_value = '2016-01-19T20:21:22.222425'

datetime\_value = datetime.strptime(string\_value, '%Y-%m-%dT%H:%M:%S.%f')

strptime() използва същите командни символи, както и strftime().

Ако Ви е необходима работа само с дати (без час), можете да използвате datetime.date.

from datetime import date

print(date(2016, 1, 19))

d = date.today()

print(d)

from datetime import datetime

dt = datetime.now()

d = dt.date() # създава нов обект date, съдържащ само информацията за дата от обект datetime

print(d)

### Аритметика с време

**!!!! Предупреждение** Никога не смятайте на ръка стойности с дати и часове - винаги използвайте специализираните механизми за това

from datetime import datetime , timedelta

d = datetime.now()

t = timedelta(hours=2)

print(d + t)

print(d - t)

print(t)

from datetime import datetime , timedelta

d = datetime.now()

d\_earlier = datetime(2016, 1, 2, 18, 15)

td = d - d\_earlier

print(td)

print(td.days)

print(td.seconds)

print(td.microseconds)

print(td.total\_seconds())

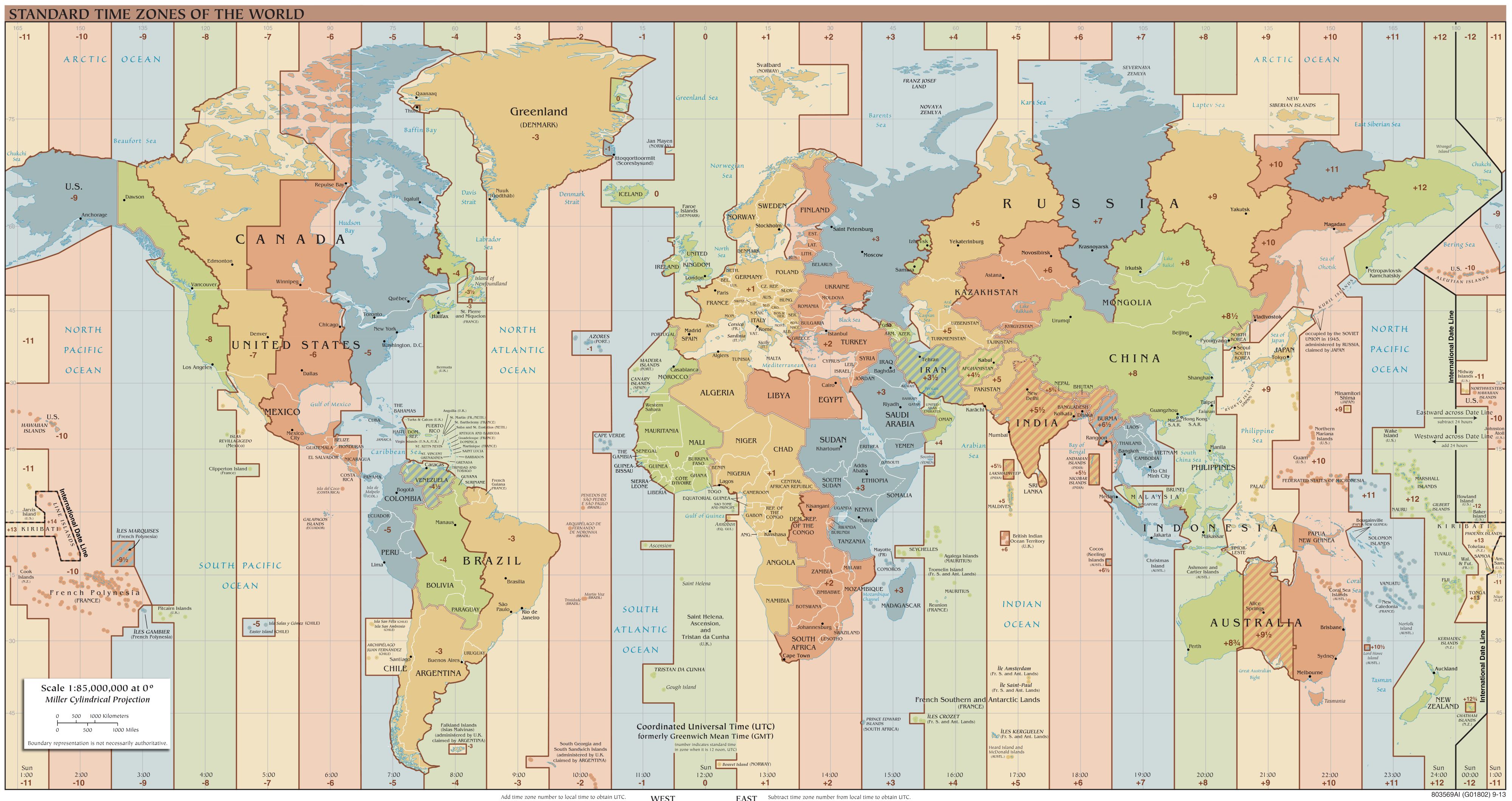
### [Задача: 2. Намерете времето с най-големи продажби](http://python3.softuni.bg/student/lecture/assignment/569e362f6e8efb1652742e99/?back=%2Fstudent%2Flecture%2F56801568131b1642fba73799%2F)

## Часови зони, DST и други особености

### Часови зони

Първоначално времето се е отчитало спрямо зенита на Слънцето за съответното място (най-високата точка на Слънцето в продължение на деня). Поради (почти) сферичната форма на Земята на различните места зенитът се случва по различно време. За да се стандартизира отчитането на времето се въвеждат часовите зони.

[Wikipedia - Time zone](https://en.wikipedia.org/wiki/Time_zone)



[Часови зони (пълен размер) - от Wikipedia](http://python3.softuni.bg/media/lectures/lecture_05/Standard_World_Time_Zones.png)

Часовите зони зависят както от локацията на мястото, така и от решения на всяка държава спрямо кои часови зони ще отчита времето си. Както се вижда на картата по-горе, часовите зони въобще не отговарят на меридианите.

Във видеоклиповете ще намерите подробни и интересни обяснения за часовите зони:

[The Problem with Time & Timezones - Computerphile](https://www.youtube.com/watch?v=-5wpm-gesOY)

[Strangest Time Zones of the World](https://www.youtube.com/watch?v=uW6QqcmCfm8)

### Daylight saving time

[Daylight Saving Time Explained](https://www.youtube.com/watch?v=84aWtseb2-4)

### Високосни неща

Високосните години се добавят, за да компенсират факта, че Земята се завърта около Слънцето за малко повече от 365 дни.

[What is a Leap Year?](https://www.youtube.com/watch?v=xX96xng7sAE)

По същата причина към времето се добавят и високосни секунди. За разлика от високосните години, които следват предварително установен график, високосните секунди се добавят на база много по-точни измервания, и не са фиксирани.

[Wikipedia - Leap year](https://en.wikipedia.org/wiki/Leap_year)

[Wikipedia - Leap second](https://en.wikipedia.org/wiki/Leap_second)

[SciShow - A Brief History of Timekeeping (с обяснение защо е необходима високосна секунда)](https://www.youtube.com/watch?v=URK9Z2G71j8)

### От гледна точка на програмиста

Винаги, когато имате възможност, използвайте стандартното форматиране на дати и часове - [ISO8601](https://en.wikipedia.org/wiki/ISO_8601).

from datetime import datetime

import pytz

z = pytz.timezone('Europe/Sofia')

d = datetime.now(tz=z)

print(d.isoformat())

# 2016-01-19T20:23:14.608574+02:00

import iso8601

d = iso8601.parse\_date('2016-01-19T20:23:14.608574+02:00')

print(d)

print(d.isoformat())

# 6. Обектно ориентирано програмиране с Python

[Screencast от лекцията](http://python3.softuni.bg/media/Lecture06.mp4)

## Какво е клас и обект?

Обектът е конструкция, обединяваща в себе си данни и методите за работа с тях.

from datetime import datetime

d = datetime.now()

# полета / атрибути / данни

d.year

d.hour

# методи

d.isoformat()

d.replace(hour=0)

Класът е дефиниция или шаблон, по който се създават обекти.

## ООП в Python

### Дефиниране на клас

class Figure:

pass

figure = Figure()

Конвенцията за именоване на класове в Python е имената на класовете да започват с главна буква, и да са CamelCase - също както в Java & C#.

### Методи

Методите са функции, които са асоциирани към обектите, и (обикновено) работят с данните в тях.

from datetime import datetime

d1 = datetime.now()

d1.isoformat() # isoformat() е метод на обекта d1

d2 = datetime(2015, 1, 21, 18, 21, 22)

d2.isoformat()

Методите се дефинират като част от дефиницията на класовете:

class Figure:

def print(self):

print("Figure")

figure = Figure()

figure.print()

Методите задължително трябва да имат поне един параметър, който е обектът, чрез който е извикан метода. В други езици това еthis, но в Python трябва да се опише изрично. Името на този параметър може да е произволно, но по конвенция е прието да бъдеself.

Всичко, което важи за функции, важи и за методи - параметри (позиционни и keyword), default стойности, \*args и \*\*kwargs.

Конвенцията за именоване на методи в Python е същата, както и за обикновени функции - "snake\_case" - имената са изцяло "lowercase" , и разделителят между думите е "долно тире" / "подчертавка" - \_\_.

#### Видимост на методи

Всички методи, дефинирани в класа, са видими и извън него - т.е. са public.

Конвенцията за дефиниране на "private" методи е името на метода да започва с \_ - например \_prepare\_data().

#### Конструктор

Конструктор на клас в Python се дефинира със специалния метод \_\_init\_\_()

class Figure:

def \_\_init\_\_(self, position, color):

print("Position: ", position)

print("Color: ", color)

figure = Figure((0, 0), 'black') # параметрите, подадени при създаване на инстанция, се предават на \_\_init\_\_()

### Атрибути (полета)

class Figure:

def \_\_init\_\_(self, center\_x, center\_y, color='black'):

self.center\_x = center\_x

self.center\_y = center\_y

self.color = color

Стандарт в Python е атрибутите да се дефинират в конструктора \_\_init\_\_ (). Въпреки, че в Python е възможно да дефинирате нов атрибут навсякъде в програмата си, стремете се да се придържате към дефинирането им само в конструктора.

Конвенцията за именоване на атрибути в Python е същата, както и за обикновени променливи - "snake\_case" - имената са изцяло "lowercase" , и разделителят между думите е "долно тире" / "подчертавка" - \_\_.

#### Видимост на полета

Всички полета, дефинирани в класа, са видими и извън него - т.е. са public.

Конвенцията за дефиниране на "private" полета е името на полето да започва с \_ - например \_color.

class Figure:

def \_\_init\_\_(self, center\_x, center\_y, color='black'):

self.\_center\_x = center\_x

self.\_center\_y = center\_y

self.\_color = color

# полетата \_color, \_center\_x и \_center\_y са видими, но не бива да се достъпват директно.

### Наследяване и override на методи

class Figure:

def \_\_init\_\_(self, center\_x, center\_y, color='black'):

self.center\_x = center\_x

self.center\_y = center\_y

self.color = color

def scale(scale\_factor):

# nothing to do here - scaling does not affect center coordinates

pass

class Circle(Figure):

def \_\_init\_\_(self, center\_x, center\_y, radius, color='black'):

super().\_\_init\_\_(center\_x, center\_y, color)

self.radius = radius

def scale(scale\_factor):

super().scale(scale\_factor) # it is a GOOD idea to always call the super() method

self.radius = self.radius \* scale\_factor

Наследяването се прави, като при дефиниране на класа в скоби след името се поставят родителските класове, от които новият клас трябва да наследи.

Python поддържа множествено наследяване, и благодарение на спецификите на реализацията на обекти и класове в езика, множественото наследяване работи много добре, и се използва широко.

За да извикате родителския метод, в Python e необходимо да използвате super() - **обърнете внимание на скобите**.

### Специални методи

В Python има немалко специални методи, които можете да предефинирате. Имената на тези методи са оградени с двойни долни тирета - пример за това е конструкторът \_\_init\_\_.

Най-често използваният специален метод, освен конструктора, е \_\_str\_\_ - еквивалентът на toString() в Java и ToString() в C#.

class Figure:

def \_\_init\_\_(self, center\_x, center\_y, color='black'):

self.center\_x = center\_x

self.center\_y = center\_y

self.color = color

def \_\_str\_\_(self):

return "Figure - center\_x={} , center\_y={}, color={}".format(

self.center\_x,

self.center\_y,

self.color

)

f = Figure(10, 20, 'red')

print(str(f))

print(f)

print("This is a {}".format(f))

В Python има и още един стандартен метод с подобно предназначение - \_\_repr\_\_ (съкращение от representation - представяне на обекта).

Обикновено \_\_repr\_\_ се използва за техническо представяне на обекта като string, като в него е добре да се включи цялата информация за обекта, която да помогне на програмиста да се ориентира за състоянието на обекта. Методът \_\_str\_\_ се използва за представяне на обекта на потребителя - обикновено той включва "човешко" форматиране на данните, локализация и други. Ако в клас е дефиниран само един от тези два метода, Python автоматично ще използва дефинираният метод, когато е необходимо.

Други специални методи могат да бъдат предефинирани в класа, за overload на оператори, контрол на достъп до атрибути на обектите и много други.

class Figure:

def \_\_init\_\_(self, center\_x, center\_y, color='black'):

self.center\_x = center\_x

self.center\_y = center\_y

self.color = color

def \_\_eq\_\_(self, other):

return (

self.center\_x == other.center\_x

and self.center\_y == other.center\_y

and self.color == other.color

)

...

f = Figure(10, 20, 'red')

f2 = Figure(10, 20, 'red')

...

print(f == f2)

Пълно описание на тези специални методи можете да намерите в [документацията на Python](https://docs.python.org/3/reference/datamodel.html#basic-customization) .

## Ползите от ООП

ООП е една от основните парадигми в програмирането. [Wikipedia - OOP](https://en.wikipedia.org/wiki/Object-oriented_programming)

Възникнала преди повече от 60 години, в зората на компютърната индустрия, и до днес ООП продължава да заема централно място под една или друга форма в модерните езици за програмиране.

По-долу са обяснени само някои от предимствата, които ни дава този подход на изграждане на софтуерни програми.

### Енкапсулация

Една от най-големите ползи от ООП е възможността да "скрием" детайлите на реализацията от хората, които ще използват класовете, които създаваме. При това основният момент не е самото "криене", а това другите хора и класове да не се занимават и да не се налага да знаят детайлите за вътрешното представяне на информацията в обекта.

Това, от една страна улеснява много използването на нашия код, а от друга дава възможност да променяме вътрешното представяне на информацията, без да е необходимо останалата част от кода да бъде променяна.

Вторият аспект на енкапсулацията е, че имаме възможност да контролираме достъпа и съдържанието на данните, с които работи обекта.

### Полиморфизъм

Чрез наследяването на класове, имаме възможност да създадем базов клас, които дефинира поведение (методи) и данни, и наследници на този клас, които реализират специфични функционалности и разширяват базовия клас.

Полиморфизмът ни дава възможност да работим с обекти от наследяващите класове, като използваме само базовите методи, без да се интересуваме на какви класове конкретно са инстанции те.

В примера по-долу ще покажем нагледен пример за полиморфизъм.

## Пример

Нека се опитаме на превърнем това:

[

{"type": "square", "center\_x": 0, "center\_y": 0, "side": 2, "color": "black"},

{"type": "square", "center\_x": 0, "center\_y": 0, "side": 100, "color": "red"},

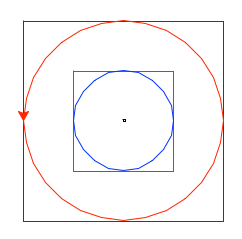
{"type": "square", "center\_x": 0, "center\_y": 0, "side": 200, "color": "blue"},

{"type": "circle", "center\_x": 0, "center\_y": 0, "radius": 50, "color": "blue"},

{"type": "circle", "center\_x": 0, "center\_y": 0, "radius": 100, "color": "red"}

]

в това:



## Полезни функции

Тези стандартни функции в Python ни дават възможност да получим информация за обектите по време на изпълнение на програмата.

### isinstance

Проверка дали обект е инстанция на даден клас

from datetime import datetime, date

d = date.today()

print(isinstance(d, date)) # True

print(isinstance(d, datetime)) # False

### hasattr, setattr и getattr

from datetime import datetime, date

d = date.today()

dt = datetime.now()

hasattr(d, 'year') # True

hasattr(dt, 'year') # True

hasattr(d, 'hour') # False

hasattr(dt, 'hour') # True

...

print(getattr(d, 'year')) # 2016

print(getattr(d, 'hour')) # will raise an AttributeError

print(getattr(d, 'hour', 'no hour is defined in d')) # default value if the attribute does not exist

### issubclass

issubclass(date, datetime) # False

issubclass(datetime, date) # True

draw\_functions.py

import sys

import json

import turtle

from figures import draw\_circle, draw\_square

def main():

if len(sys.argv) < 2:

print("Usage: {} input-file.json".format(sys.argv[0]))

return 1

try:

input\_data = load\_input\_data(sys.argv[1])

draw\_figures(input\_data)

except Exception as e:

print("Invalid input file provided! Error: " + str(e))

return 2

def load\_input\_data(input\_filename):

with open(input\_filename) as f:

input\_data = json.load(f)

return input\_data

def draw\_figures(figures\_info):

t = turtle.Turtle()

t.speed('fast')

for f\_info in figures\_info:

figure\_type = f\_info['type']

if figure\_type == 'square':

draw\_square(

turtle\_instance=t,

center\_x=f\_info['center\_x'],

center\_y=f\_info['center\_y'],

side=f\_info['side'],

color=f\_info['color']

)

elif figure\_type == 'circle':

draw\_circle(

turtle\_instance=t,

center\_x=f\_info['center\_x'],

center\_y=f\_info['center\_y'],

radius=f\_info['radius'],

color=f\_info['color']

)

else:

raise ValueError("Unsupported figure")

turtle.exitonclick()

if \_\_name\_\_ == "\_\_main\_\_":

sys.exit(main())

figures.py

def draw\_circle(turtle\_instance, center\_x, center\_y, radius, color):

turtle\_instance.penup()

turtle\_instance.goto(center\_x - radius, center\_y) # From docs: The center is radius units left of the turtle;

turtle\_instance.pendown()

turtle\_instance.color(color)

turtle\_instance.circle(radius)

def draw\_square(turtle\_instance, center\_x, center\_y, side, color):

half\_side = side / 2

left = center\_x - half\_side

top = center\_y + half\_side

turtle\_instance.penup()

turtle\_instance.goto(left, top)

turtle\_instance.pendown()

turtle\_instance.color(color)

turtle\_instance.forward(1)

turtle\_instance.setheading(270) # point the turtle down

for \_ in range(4):

turtle\_instance.forward(side)

turtle\_instance.left(90)

#### **Задача: 3. Анализ на данни от верига магазини**

[Назад](http://python3.softuni.bg/student/lecture/568015bf131b1642faa73799/)

Разполагаме с каталог на стоки и данни за продажби на голям производител на спортни стоки, и трябва да направим анализ на тази информация.

* [Архив, съдържащ всички файлове](http://python3.softuni.bg/media/lectures/lecture_07/sales-analysis-assignment-all.zip)
* Отделни файлове:
  + [Архив, съдържащ продуктовия каталог](http://python3.softuni.bg/media/lectures/lecture_07/catalog.csv.zip)
  + [Архив, съдържащ извадка от 10'000 продажби](http://python3.softuni.bg/media/lectures/lecture_07/sales-10K.csv.zip)
  + [Архив, съдържащ извадка от 100'000 продажби](http://python3.softuni.bg/media/lectures/lecture_07/sales-100K.csv.zip)
  + [Архив, съдържащ извадка от 1'000'000 продажби](http://python3.softuni.bg/media/lectures/lecture_07/sales-1M.csv.zip)

Крайната цел на задачата е да анализираме данните и да визуализираме:

$ python3 analyze.py catalog.csv sales-10K.csv

Обобщение

---------

Общ брой продажби: 10000

Общо сума продажби: 3191507.82 €

Средна цена на продажба: 319.150782 €

Начало на период на данните: 2015-12-01 07:00:48+00:00

Край на период на данните: 2016-01-24 20:49:38+00:00

Сума на продажби по категории (top 5)

-----------------------------

Якета : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1250.23 €

Обувки за футбол : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 912.30 €

Екипи : \*\*\*\*\*\*\*\*\*\*\*\*\*\* 513.45 €

Топки : \*\*\*\*\*\*\*\*\*\*\*\* 502.52 €

T-SHIRTS : \*\*\*\*\*\*\*\*\* 420.19 €

Сума на продажби по градове (top 5)

---------------------------

Берлин : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1250.23 €

Франкфурт : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 912.30 €

Париж : \*\*\*\*\*\*\*\*\*\*\*\*\*\* 513.45 €

Марсилия : \*\*\*\*\*\*\*\*\*\*\*\* 502.52 €

Виена : \*\*\*\*\*\*\*\*\* 420.19 €

Часове с най-голяма сума продажби (top 5)

-----------------------------------------

2016-02-20 20:00 : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1250.23 €

2016-02-20 18:00 : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 912.30 €

2016-02-20 22:00 : \*\*\*\*\*\*\*\*\*\*\*\*\*\* 513.45 €

2016-02-20 21:00 : \*\*\*\*\*\*\*\*\*\*\*\* 502.52 €

2016-02-20 19:00 : \*\*\*\*\*\*\*\*\* 420.19 €

Обърнете внимание, че часът от последния анализ трябва да бъде в UTC часова зона - "+00:00".

Звездичките играят роля на **bar-chart с хоризонтални bars** - колкото е по-голяма стойността, толкова са повече звездичките, като максималният брой звездички е 30 (или подобно число), за да могат да се съберат на нормален конзолен екран.

### Входни данни и описание на формат на данните

#### Продуктов каталог - файл 'catalog.csv'

Файловете с продуктовия каталог са във формат CSV (UTF-8), и съдържа следните колони:

1. Идентификационен номер на артикула;
2. Наименование на артикула;
3. Цветове, в които артикулът е наличен;
4. Група на артикула;
5. Спорт, за който е предназначен артикулът;
6. Категория
7. Подкатегория
8. Пол, за който е предназначен артикула - мъже, жени, unisex, деца, бебета

Примерни редове от файла с каталога:

"538352","TYGUN II","BLACK/RUNNINWHT","FOOTWEAR","BOXING","SHOES","SHOES (LOW)","Men"

"42259","M TEAM TEE","WHITE/DARKNAVY","TEXTILES","TENNIS","T-SHIRTS","T-SHIRT (SHORT SLEEVE)","Men"

"562319","NOVA WALK 06","DARONX/METSIL/DSHALE","FOOTWEAR","WALKING","SHOES","SHOES (LOW)","Men"

"G13130","COMPOUND","BLACK1/BLACK1/LGTIPD","FOOTWEAR","GOLF","SHOES","SHOES (LOW)","Men"

"396559","+TG SWERVE DBMI","WHITE/BLACK","HARDWARE","FOOTBALL/SOCCER","BALLS","BALL (MACHINE-STITCHED)","Men"

"34549","LTA SS2G C","RUNWHI/SHABLU/VAPOUR","FOOTWEAR","BASKETBALL","SHOES","SHOES (LOW)","Kid"

#### Данни за продажби - файлове 'sales-\*.csv'

Файловете с продажби са във формат CSV (UTF-8), и съдържат следните колони:

1. Идентификационен номер на артикула;
2. Държава, в която е била извършена продажбата (ISO code)
3. Име на град, в която е била извършена продажбата;
4. Дата/час на продажбата с timezone, във формат ISO8601;
5. Цена на продажбата (цените на един и същ артикул в различните държави са различни)

Примерни редове от файл с продажби:

"561712","ES","Murcia","2015-12-11T17:14:05+01:00",43.21

"K81938","FR","Nantes","2016-01-14T20:58:38+01:00",36.57

"41975","IT","Catania","2016-01-12T10:57:50+01:00",409.58

"538352","DE","Düsseldorf","2015-12-18T20:50:21+01:00",95.03

### Насоки

##### **Parse на дати в ISO8601 формат**

За да parse-нете датите във файловете с продажби, използвайте модула iso8601

import iso8601

...

dt = iso8601.parse\_date('2015-12-11T17:14:05+01:00')

print(dt)

2015-12-11 17:14:05+01:00

##### **Работа с datetimes, които съдържат информация за часови зони**

Вземане на текуща дата/час в часова зона UTC

from datetime import datetime, timezone

now\_utc = datetime.now(tz=timezone.utc)

print(now\_utc)

'2016-01-26 14:38:02.720755+00:00'

##### **Конвертиране към от друга часова зона към UTC**

from datetime import datetime, timezone

import iso8601

d = iso8601.parse\_date('2015-12-18T20:50:21+01:00')

print(d)

'2015-12-18 20:50:21+01:00'

d\_in\_utc = d.astimezone(timezone.utc)

print(d\_in\_utc)

'2015-12-18 19:50:21+00:00'

import sys

import os

import csv

import operator

from pprint import pprint

from product import Product

from sale import Sale

def main():

if len(sys.argv) < 3:

print("You need to add catalog as first argument and sales as second in csv format!")

return 1

#try:

product\_catalog = load\_product\_catalog(sys.argv[1])

sales = load\_sells(sys.argv[2])

print("Обобщение:\n-----------------------------")

print("\tОбщ брой продажби: {}".format(len(sales)))

sell\_sum = sum(float(sale.price) for sale in sales)

print("\tОбщо сума продажби: {:.2f} €".format(sell\_sum))

avg\_sell = sell\_sum / len(sales)

print("\tСредна цена на продажба: {:.6f} €".format(avg\_sell))

sell\_with\_min\_datetime = min(sales, key=lambda s: s.sell\_datetime)

print("\tНачало на период на данните: {}".format(sell\_with\_min\_datetime.sell\_datetime))

sell\_with\_max\_datetime = max(sales, key=lambda s: s.sell\_datetime)

print("\tКрай на период на данните: {}".format(sell\_with\_max\_datetime.sell\_datetime))

sales\_by\_category = {}

sales\_by\_city = {}

sales\_by\_datetime = {}

for sale in sales:

sale\_category = product\_catalog[sale.product\_id].category

if sale\_category in sales\_by\_category:

sales\_by\_category[sale\_category] += sale.price

else:

sales\_by\_category[sale\_category] = sale.price

if sale.city in sales\_by\_city:

sales\_by\_city[sale.city] += sale.price

else:

sales\_by\_city[sale.city] = sale.price

sale\_date = sale.sell\_datetime.strftime("%Y-%m-%d %H:00")

if sale\_date in sales\_by\_datetime:

sales\_by\_datetime[sale\_date] += sale.price

else:

sales\_by\_datetime[sale\_date] = sale.price

sorted\_products\_by\_category = sorted(sales\_by\_category.items(), key=operator.itemgetter(1), reverse=True)

print("Сума на продажби по категории (top 5)")

print("-----------------------------")

print\_top\_five(sorted\_products\_by\_category)

sorted\_products\_by\_city = sorted(sales\_by\_city.items(), key=operator.itemgetter(1), reverse=True)

print("Сума на продажби по градове (top 5)")

print("-----------------------------")

print\_top\_five(sorted\_products\_by\_city)

sorted\_products\_by\_datetime = sorted(sales\_by\_datetime.items(), key=operator.itemgetter(1), reverse=True)

print("Часове с най-голяма сума продажби (top 5)")

print("-----------------------------")

print\_top\_five(sorted\_products\_by\_datetime)

def print\_top\_five(elements):

max\_price = elements[0][1]

for key, value in elements[:5]:

print("\t{} : ".format(key) + "\*" \* int(30 / (max\_price / value)) + " {:.2f} €".format(value))

def load\_product\_catalog(file):

if os.access(file, os.R\_OK) and os.path.isfile(file):

products = {}

with open(file) as f:

lines = csv.reader(f, dialect='excel')

for line in lines:

id = line[0]

product = Product(\*line)

products[id] = product

return products

else:

raise ValueError("Inaccessible file '{}'".format(file))

def load\_sells(file):

if os.access(file, os.R\_OK) and os.path.isfile(file):

sales = []

with open(file) as f:

lines = csv.reader(f, dialect='excel')

for line in lines:

sale = Sale(\*line)

sales.append(sale)

return sales

else:

raise ValueError("Inaccessible file '{}'".format(file))

if \_\_name\_\_ == "\_\_main\_\_":

sys.exit(main())

class Product:

def \_\_init\_\_(self, id, name, colors, group, sport, category, subcategory, gender):

self.id = id

self.name = name

self.colors = colors

self.group = group

self.sport = sport

self.category = category

self.subcategory = subcategory

self.gender = gender

import iso8601

from datetime import datetime, timezone

class Sale:

def \_\_init\_\_(self, product\_id, country, city, sell\_datetime, price):

self.product\_id = product\_id

self.country = country

self.city = city

self.sell\_datetime = iso8601.parse\_date(sell\_datetime)

self.sell\_datetime = self.sell\_datetime.astimezone(timezone.utc)

self.price = float(price)

# 8. Фокусите в Python

В Python имаме на разположение приличен арсенал от "фокуси", които правят програмирането на езика по-лесно и приятно. Фокусите включват както "syntactic sugar" (улеснения в синтаксиса), така и полезни класове и функции в стандартната библиотека.

За да демонстрираме всичко в тази лекция, ще използваме кода от [задачата в предишната лекция - Анализ на данни от верига магазини](http://python3.softuni.bg/student/lecture/assignment/56a4ab616e8efb456bd29b06/) .

## ВАЖНО

Описаните в тази лекция концепции и похвати **не са задължителни, за да можете да работите с Python**.

Ако не се чуствате сигурни, или прецените, че не можете да разберете в детайли всичко, спокойно можете да пропуснете тази лекция.

Целта на лекцията е да хвърлим бегъл поглед "под чаршафите" на Python, и да видим как работят някои от основните му концепции.

## Cheat-sheet - полезни похвати

По-долу са дадени накратко похвати, които можете да използвате, без да е необходимо да разбирате как точно работят. В следващите секции нещата са описани в детайли.

##### **default стойности**

ако a\_list е непразен списък, ще запази стойността си; ако е None, новата стойност ще бъде празен списък:

a\_list = a\_list or []

Oбобщение на горното: присвояване на default стойности на променливи - работи за почти всички типове данни - str, int, float, dict, list, tuple, set, ....

a\_value = a\_value or DEFAULT\_VALUE

##### **list comprehension**

Създаване на списък от нещо, което можем да обиколим с for цикъл (т.нар. iterable):

[value\_expression for value in iterable]

[n\*2 for n in (1, 2, 3, 4, 5)]

[n for n in range(5)]

[0, 1, 2, 3, 4]

[n\*\*2 for n in range(5)] # n\*\*2 - n на степен 2

[0, 1, 4, 9, 16]

with open('file.txt') as f:

lines\_with\_stripped\_whitespaces = [l.strip() for l in f]

Създаване на филтриран списък от нещо, което можем да обиколим с for цикъл (т.нар. iterable):

[n for n in range(10) if n % 2 == 0]

[0, 2, 4, 6, 8]

[n\*\*2 for n in range(10) if n % 2 == 0] # n\*\*2 - n на степен 2

[0, 4, 16, 36, 64]

with open('file.txt') as f:

lines\_with\_stripped\_whitespaces\_non\_empty = [l.strip() for l in f if l.strip()]

##### **dict comprehension**

Подобно на list comprehension, създаваме dict по нещо, което можем да обиколим с for цикъл (т.нар. iterable):

{key\_expression: value\_expression for value in iterable}

{n: n\*2 for n in (1, 2, 3, 4, 5)}

Важно: между "ключ" и стойност трябва да има :, както при дефиниране на dict

{n: n\*\*2 for n in range(10)}

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16} # ключ: числото , стойност: квадрат на числото

with open('file.txt') as f:

lines\_by\_line\_number = {line\_number+1: l.strip() for line\_number, l in enumerate(f)}

## iPython

Python има интерактивен shell - REPL . Когато стартирате Python без параметри, ще видите нещо подобно на това:

Python 3.4.2 (default, Dec 30 2014, 18:34:07)

[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>

всеки ред, който напишете, ще бъде изпълнен от интерпретатора на Python, резултатът ще бъде отпечатан. Това е много полезно, когато искате бързо и лесно да проверите как работи нещо.

Проблемът е, че по обикновеният REPL на Python няма смислен help, няма autocompletion, history, което го прави неудобен.

iPython е алтернативен REPL, който има гореспоменатите функционалности, както и много други, което го прави изключително удобен, ако искате лесно и бързо да проверите как и дали работи нещо.

### инсталиране

pip3 install ipython

Ако сте **под Windows**, трябва да инсталирате и още един пакет:

pip3 install pyreadline

Стартирате го с ipython3 или само ipython, в зависимост от OS

Python 3.4.2 (default, Dec 30 2014, 18:34:07)

Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.

? -> Introduction and overview of IPython's features.

%quickref -> Quick reference.

help -> Python's own help system.

object? -> Details about 'object', use 'object??' for extra details.

In [1]:

iPython има много силен и удобен code completion - напишете име на променлива, . и след това натиснете клавиш TAB.

DEMO

## Булеви оператори

Често се налага да сложим "default" стойност на променлива, ако тя към момента е празна

def print\_list(list\_to\_print):

for e in list\_to\_print:

print(e)

Ако обаче по някаква причина на тази функция подадем стойност None, ще се получи грешка.

list\_of\_sales = None

... # поради грешка сме пропуснали да инициализираме списъка

print\_list(list\_of\_sales)

TypeError Traceback (most recent call last)

1 def print\_list(list\_to\_print):

----> 2 for e in list\_to\_print:

3 print(e)

4

TypeError: 'NoneType' object is not iterable

**Един от начините**, по който можем да си осигурим, че параметърът няма да има стойност None, когато започнем да го обикаляме, е :

def print\_list(list\_to\_print):

list\_to\_print = list\_to\_print or [] # <<<<<

for e in list\_to\_print:

print(e)

или малко по-краткото:

def print\_list(list\_to\_print):

for e in list\_to\_print or []: # <<<<<

print(e)

#### Как работи това?

Примерът по-горе работи благодарение на следните концепции, описани детайлно по-долу:

1. "truthy" & "falsy" стойности
2. minimal/short-circuit evaluation
3. резултатът от булев израз е последно проверената/изпълнената стойност;

##### **"truthy" & "falsy" стойности**

Python използва концепцията за "truthy" & "falsy", подобно на много други скриптови езици. Ако променлива се използва в булев контекст - част от if или с boolean оператори, в зависимост от типа на стойността Python **третира стойността като boolean** - True или False, но без да я конвертира в boolean.

Примери за **Falsy** стойности в зависимост от типа:

* **int** - 0
* **float**: 0.0 (но трябва да се внимава с това, заради грешки в точността на представяне на float числата, особено ако float стойността е резултат от изчисления)
* **str** - "" (празен стринг)
* **bytes** - b"" (празен bytes обект)
* **list**, **tuple**, **dict**, **set** - празни
* None
* False

Всички останали стойности за съотвения тип се третират като **Truthy**, например:

* **int** - 1, -1, -5237922296986728930672677403958
* **float**: 0.000000000001, 320573.3245
* **str** - " " (стринг, съдържащ интервал), "any other string value"
* **bytes** - b" " (bytes, съдържащ интервал),b"\x32bytes\x62"
* непразни **list**, **tuple**, **dict**, **set** - [" "], [0], [1,2,3,4, "a"], {"key": "value"}, {2, 4, 6, 8}, ("", "")
* True
* Обект

Това ни дава възможност да проверяваме:

a\_list = []

...

if a\_list:

print("list is not empty")

if not a\_list:

print("the list is empty")

##### **minimal/short-circuit evaluation**

Другата причина горните примери да работят е [**minimal evaluation** / **short-circuit evaluation**](https://en.wikipedia.org/wiki/Short-circuit_evaluation) - в един булев израз се изпълняват само нещата, които има смисъл да се изпълнят, за да се определи резултата от израза.

За логическо "или/or" е достатъчно само едната стойност да е True, за да бъде крайният резултат True.

if check\_condition\_1() or check\_condition\_2():

...

Ако първата функция върне True, втората няма смисъл да се изпълнява, тъй като крайният резултат ще бъде True, независимо какво ще върне втората функция.

От примерната функция по-горе имахме използване на булев оператор or така: list\_to\_print = list\_to\_print or [].

* ако list\_to\_print е не-празен списък, тогава той ще се определи като **Truthy** стойност, и това ще бъде резултатът от целия израз, който ще се присвои отново на променливата;
* ако list\_to\_print е празен списък или None ще бъде определен като **Falsy**, и интерпретаторът ще трябва да изпълни и втората стойност, която ще бъде и крайният резултат от изпълнението на израза.

## unpacking

Лесен и бърз начин да присвоим няколко стойности на няколко променливи едновременно е да използваме tuple unpacking.

x, y = 100, 200 # присвоява стойностите на от tuple (100, 200) на двете променливи x & y

print(x)

print(y)

Имайте предвид, че ако броят на стойностите в tuple от двете страни не е един и същ, ще получите грешка:

x, y = 100, 200, 300

ValueError Traceback (most recent call last)

----> 1 x, y = 100, 200, 300

ValueError: too many values to unpack (expected 2)

Ако желаем да присвоим първите няколко стойности на променливи, и всички останали да игнорираме, можем да използваме:

x, y, \*other\_values = 100, 200, 300, 400

print(other\_values) # other\_values е list с всички стойности, които не са били присвоени

Ако няма да използваме останалите стойности, конвенцията е да се използва \_ като име на променливата :

x, y, \*\_ = 100, 200, 300, 400

## generators

Генераторите са механизъм, при който функция може да връща резултати един по един, като "заспива" между отделните връщания на стойности.

def range\_equivalent(start, end=None, step=None):

step = step if step is not None else 1

value = start

while end is None or value < end:

yield value

value += step

Използване на генератори:

for n in range\_equivalent(2, 5):

print(n)

2

3

4

#### Приложение на генераторите

Често генератори се използват, когато не е необходимо да създадете всички стойности наведнъж.

Например, ако трябва да обработим данните от файл, е доста вероятно да не е добре да прочитаме целия файл в паметта, защото:

* файлът може да се окаже прекалено голям, и да не може да се помести в паметта;
* възможно е по средата на файла да преценим, че няма нужда да продължаваме - например ако сме намерили каквото търсим; в този случай сме прочели от диска само толкова данни, колкото сме обработили - не целия файл;

Advanced:

Всъщност range\_equivalent не е генератор, а е **generator function**. Когато в тялото на функция има ключовата дума yield, при извикване на функцията Python създава обект generator, който се грижи да изпълнява и прекъсва изпълнението на тялото на функцията.

Проблясък:

Comprehensions, които разгледахме по-горе, са всъщност генератори.

DEMO

## Полезни вградени функции

* min(iterable или args) & max(iterable или args)
* sum(iterable)
* all(iterable) & any(iterable)
* enumerate()

## lambda функции

lamda функциите са мини-функции, които се състоят само от 1 statement. Използват се много често

### Някои приложения на lambda функциите

#### сортиране

a\_list = [ (4, 'b'), (2, 'a'), (1, 'c'), (3, 'f') ]

a\_list.sort()

print(a\_list)

[(1, 'c'), (2, 'a'), (3, 'f'), (4, 'b')]

a\_list.sort(key=lambda item: item[1])

print(a\_list)

[(2, 'a'), (4, 'b'), (1, 'c'), (3, 'f')]

#### map

map() прилага подадената функция върху всеки елемент от подадения iterable

DEMO

#### filter

filter() прилага подадената фукция върху всекли елемент от подадения iterable, и връща само тези елементи, за които фукцията е върнала truthy резултат.

DEMO

# 9. Работа с бази данни

## Релационни бази данни

ВАЖНО: Лекцията ще Ви даде само бегла представа за работа с релационни бази данни и SQL - създаване на таблица, вкарване, извличане, актуализиране и изтриване на данни.

За примерите в лекцията ще използване данните от предишната лекция:

* [catalog-example.csv](http://python3.softuni.bg/media/lectures/lecture_08/catalog-example.csv)
* [sales-example.csv](http://python3.softuni.bg/media/lectures/lecture_08/sales-example.csv)
* [sales-example.db](http://python3.softuni.bg/media/lectures/lecture_08/sales-example.db)

### SQLite

SQLite е малка и удобна библиотека за бази данни, която съхранява цялата база данни в един файл. Използва се на много места (например в браузърите Mozilla Firefox и Google Chrome, Android, iOS и т.н.). За работа с SQLite не е необходимо да стартирате сървър или да правите каквото и да било - просто се "свързвате" към базата данни.

Python има вградена поддръжка на SQLite, без да е необходимо да правите нищо допълнително (даже и под Windows :o) ). За целите на лекцията и задачите ще използваме SQLite.

Можете да използвате SQLite директно от Вашия терминал:

$ sqlite3 example.db

SQLite version 3.8.6 2014-08-15 11:46:33

Enter ".help" for usage hints.

sqlite>

И понеже някои от Вас не обичат Големия Страшен Черен Терминал (ГСЧТ), можете да използвате някоя програма с графичен интерфейс, за да прегледате съдържанието на SQLite база данни, например:

* [SQLite Studio](http://sqlitestudio.pl/?act=download) - работи под Windows, Mac, Linux
* [SQLite Free](https://itunes.apple.com/us/app/sqlite-free-datum/id901631046?mt=12) - работи само под MacOS
* [SQLite Administrator](http://sqliteadmin.orbmu2k.de/) - работи само под Windows
* [SQLite Manager](https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/) - addon за Mozilla Firefox, мултиплатформен
* и мнооооого мноого други :о)

За първоначалната проба ще използваме предварително подготвена база с данни от задачата с продажбите от предишната лекция.

DEMO

### Други бази данни

С другите бази данни се работи по подобен начин, както ще разгледаме в тази лекция SQLite.

За да можете да се свържете с Вашата база данни, ще трябва да инсталирате необходимия Ви драйвер с pip install ...

* MySQL
* PostgreSQL - [psycopg2](http://initd.org/psycopg/docs/install.html#use-a-python-package-manager)
* Oracle - [cx\_Oracle](http://cx-oracle.sourceforge.net/)
* MS SQL Server - [pymssql](http://pymssql.org/en/latest/intro.html#install)

Имайте предвид, че при различните бази данни SQL се различава малко - дали като типове колони, които се поддържат, дали като специфики при някои операции. Прочетете в документацията дали Вашата база данни не работи с нестандартни функционалности на SQL езика

## SQL

[SQL](https://en.wikipedia.org/wiki/SQL) е стандартният език за работа с релационни бази данни. Създаден през 1974, и до днес SQL се използва и развива.

Всяка популярна релационна база данни поддържа SQL до една или друга степен. Доста често базите данни добавят собствени разширения към езика SQL, които да отразят техните специфики.

## Основни концепции

### Таблица

Таблицата е основният елемент в релационните бази данни.

В релационните бази данни структурата на таблиците е фиксирана - колони с точно определен тип данни.

Във всяка таблица имаме редове (записи), като всеки ред съдържа стойности за колоните на таблицата.

### Primary key / PK / Първичен ключ

Уникално идентифицира записа в таблицата. Най-добре е да е int / integer.

Обикновено е autoincrement / serial - при вмъкване на данни в таблицата, базата данни автоматично попълва последователен int.

### Релации и Foreign key / FK / Ключ към друга таблица

Най-голямата сила на релационните бази данни е моделирането на данните като релации между таблиците.

TODO

## Параметри

!!! ВАЖНО !!! Никога не изпълнявайте SQL заявки като конкатенирате / форматирате директно стойностите в SQL стринга. Това е една от най-често срещаните грешки в модерните софтуерни системи, и се нарича [SQL injection](https://en.wikipedia.org/wiki/SQL_injection).**Винаги подавайте параметрите на SQL заявката отделно.**

TODO

## SQL: Select

select \* from catalog;

select \* from catalog where category = 'SHOES';

## SQL: Създаване на таблица

create table if not exists catalog (

id INTEGER PRIMARY KEY AUTOINCREMENT,

item\_key varchar(200),

category varchar(200)

);

## SQL: Insert

insert into catalog (item\_key, category) values ('999999', 'Some category');

## SQL: Update

update sale set city = 'Frankfurt' where city = 'Frankfurt am Main';

## SQL: Delete

delete from sale where city = 'London';

# 10. HTTP; Задачи

## Какво е HTTP?

[HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) е текстов протокол за пренос на информация, чрез който работи web, както и много услуги и системи в Интернет. Създаден през '90-те години, основната идея на HTTP остава непроменена и до днес. Актуалната, и най-често използвана версия на протокола в момента е 1.1.

Протоколът се основава на просто взаимодействие между клиент и сървър, които комуникират с текстови команди, което прави протокола използваем от практически всякакви платформи, и лесно четим. **Клиентът** изпраща заявка, а **сървърът** връща отговор на заявката.

В примерите по-долу: ⏎ - край на ред - \n

#### Заявка от клиента:

**(1)** Клиентът установява мрежова връзка от клиента до сървъра;

**(2)** Изпраща в текстов вид HTTP заявка. Най-простата HTTP заявка, валидна за версия 1.1 на протокола, е:

GET /about.html HTTP/1.1⏎

Host: python3.softuni.bg⏎

⏎

**(3)** Изчаква се отговор на заявката от сървъра

#### Отговор от сървъра:

**(4)** Сървърът връща отговор на заявката

HTTP/1.1 200 OK⏎

Server: nginx/1.6.2 (Ubuntu)⏎

Content-Type: text/html; charset=UTF-8⏎

Content-Length: 3336⏎

. . . .

⏎

<html>

<head>

<title>Таковата</title>

. . . . .

</head>

<body>

. . . . . . . . . . . . . .

</body>

</html>

**(5)** В зависимост от конфигурацията и изискванията на клиента, сървърът или прекъсва мрежовата връзка, или изчаква за последващи заявки от клиента.

## HTTP заявка

+----------------------------- Метод

|

| +---------+--------------- URI

| | |

| | | +--------- Протокол

| | | |

| | | | +---- Нов ред

v | | v v

GET /index.html HTTP/1.1⏎

|------> Host: example.com⏎

| Agent: user-agent:Mozilla/5.0 (Macintosh;...⏎

|------> Header-Name: Value

| +---> ⏎

| |

| |

| |

| Празен ред

|

Headers

### Метод

Методът указва каква операция трябва да бъде изпълнена. Най-често използваните методи са:

* GET - изисква извличане на ресурс от сървъра;
* POST - подава данни към сървъра;

Други методи, налични в HTTP, включват HEAD (за извличане само на headers на отговора, без ), OPTIONS (за извличане на мета- или допълнителна информация относно ресурса), PUT (за промяна на ресурс), DELETE (за премахване на ресурс от сървъра).

### URL

URL ([Uniform Resource Locator](https://en.wikipedia.org/wiki/Uniform_Resource_Locator)) е стандарт за представяне на адрес на ресурс.

Най-често в Web URL има следния вид:

+--------------------------- Протокол / схема

|

| +----------------+---- Host / сървър

| | |

| | |

v v v

http://python3.softuni.bg/student/lecture/56801680131b1642fba7379d/?started=1&live-view=1

^ ^ ^ ^ ^ ^

| | | | | |

| | | | | |

Ресурсен идентификатор ----+---------------------------------------+ | | | |

(или път до ресурса) | | | |

| | | |

Параметър 1 ----------------------------------------------+-------+ | |

| |

Параметър 2 --------------------------------------------------------+---------+

**URL не може да съдържа интервал**.

* Протоколът (или схемата) указва по какъв начин ще бъде достъпен този ресурс. Най-често в Web схемата е http или https;
* Host / сървър - указва адрес на сървъра, на който трябва да се достъпи ресурса
* Идентификатор на ресурса - в началото на web идентификаторът е представял директно път до файл на файловата система на сървъра; с развитието на web технологиите все по-често това е просто идентификатор, и не задължително на сървъра да съществуват физически такъв файл или директория;
* GEТ параметри - като част от URL могат да бъдат подадени допълнителни параметри, често наричани GET параметри; отделят се от иденфитикатора на ресурса с ?, и са подредени име на параметър=стойност

### Request headers

Headers представляват информация във вид име на хедър: стойност, която се изпраща заедно със заявката. С помощта на headers клиентът има възможност да укаже допълнителна информация - например какви типове информация може да обработва, какви са езиковите предпочитания, и много други.

Всеки header трябва да бъде на отделен ред. Името на header е отделено от стойността с :.

### Request body

При методите, които допускат подаване на информация към сървъра - POST, PUT, след края на request headers се подава допълнителен празен ред, след което следва съдържанието на заявката.

POST /login HTTP/1.1

Host: localhost

Content-Length: 31

username=boris&password=python3

## HTTP отговор

+------------------- Версия на HTTP

|

| +------------ Код / статус на отговора / http response code

| |

| | +-------- Текстово представяне на статуса на отговора

| | |

| | |

v v v

HTTP/1.1 200 OK⏎

+--------> Server: nginx/1.6.2 (Ubuntu)⏎

| Content-Type: text/html; charset=UTF-8⏎

| Content-Length: 3336⏎

+--------> . . . .

| +-----> ⏎

| | +--> <html>

| | | <head>

| | | <title>Таковата</title>

| | | . . . . .

| | |

| | +-->

| | |

| | Съдържание на ресурса

| |

| Празен ред - край на response headers и

| начало на съдържанието на ресурса

|

Response headers

### Код на отговор / HTTP response code

Кодът на отговора дава информация как е протекло изпълнението на заявката. Кодовете са трицифрени числа, разделени по групи според смисъла. По-долу са дадени групите , както и най-често използваните response codes

| **Група** | **Код** | **Значение** |
| --- | --- | --- |
| 1xx | Информационни статуси |  |
| 2xx |  |  |

200 | OK - заявката е изпълнена успешно, и съдържанието на ресурса е върнато в отговора | | 3xx | 301 | Moved permanently - ресурсът е бил преместен за постоянно | 302 | Found - ресурсът е открит, и трябва да бъде търсен на друг адрес | | 4xx | 400 | Bad request - заявката не е била формулирана правилно, и не е приета | 401 | Unauthorized - клиентът е оторизиран да извика тази заявка | 403 | Forbidden - клиентът няма право да изпълни тази заявка | 404 | Not found - ресурсът не е намерен | | 5xx | 500 | Възникнала е грешка на сървъра при изпълнението на заявката |

### Headers

* Content-Type - определя [MIME type / media type](https://en.wikipedia.org/wiki/Media_type) на върнатия отговор - т.е. какъв тип са върнатите данни - текст, HTML, изображение, Excel документ и т.н.
* Content-Length - размер на върнатия отговор в байтове.

### Response body

Обикновено се връща съдръжанието на ресурса, който е бил извикан, но може да бъде и празно - например при HTTP 301

Типът на съдържанието на ресурса се определя от Content-Type

## HTTP клиент с Python

В стандартната библиотека на Python има HTTP клиент, с който можете да изпълнявате HTTP заявки.

from urllib.request import urlopen

with urlopen('http://www.dir.bg') as response:

print(r.read())

DEMO

### requests

requests е външна библиотека, с която HTTP заявките се правят доста по-лесно, особено когато трябва да подавате допълнителни параметри към заявката, да изпращате файлове и т.н.

pip3 install requests

import requests

response = requests.get('http://api.fixer.io/latest?symbols=USD,GBP')

....

response = requests.get('http://api.fixer.io/latest', timeout=2.5) # timeout е в секунди

GET параметри с requests:

response = requests.get('http://api.fixer.io/latest', timeout=20, params={'symbols': 'USD,GBP'}) # timeout е в секунди

print(response.url)

## GET параметри и URL escaping

Специални символи в URL, които задължително трябва да бъдат escaped:

* ' ' - интервал
* ?
* &
* =

### Със средствата на стандартната библиотека

За да си осугирите, че str стойност ще бъде валидна част от URL, можете да използвате функцията quote

from urllib.request import quote

print(quote('Symbols ?, & and = must be escaped'))

Symbols%20%3F%2C%20%26%20and%20%3D%20%20must%20be%20escaped

### С библиотеката requests

Библиотеката requests автоматично прави escape на подадените данни.

За да изпратите GET параметри с requests, използвайте:

import requests

response = requests.get('http://api.openweathermap.org/data/2.5/weather', # основен URL

params={ # GET параметри - dict

'q': 'Sofia',

'appid': '965acdac1ae64cf06761bb563ad34d96'

},

timeout=20, # seconds

)

# От извикването по-горе, библиотеката ще композира пълния URL:

print(response.url)

http://api.openweathermap.org/data/2.5/weather?q=Sofia&appid=965acdac1ae64cf06761bb563ad34d96

# 11. Задачи и подготовка за изпит (1)

## Указания за провеждане на изпита

### Входни данни

Всички входни данни за Вашите програми по време на изпита ще се въвеждат с input().

Примерна задача: Отпечатайте подадения като входни данни string наобратно:

input\_string = input()

print(''.join(reversed(list(input\_string))))

В условието на задачата ще бъдат предоставени **примерни данни**, с които ще можете да тествате Вашето решение, но входнте данни, които ще Ви подаде Judge **ще бъдат различни от примерните данни**.

### Отваряне на файлове

Когато отваряте файлове, **ВИНАГИ ИЗПОЛЗВАЙТЕ параметър encoding='utf-8'**

with open('/path/to/file.txt', encoding='utf-8') as f:

...

Тази промяна е необходима, за да може програмата Ви да работи коректно в средата на SoftUni Judge.

### Изход / извеждане на резултатите

Бъдете изключително внимателни какво отпечатвате като резултат от Вашата програма. Както знаете, изпитът ще се проведе чрез SoftUni Judge, който ще сравнява изведените от Вашата програма резултати с предварително подготвени правилни резултати.

Напълно възможно е ако отпечатате и други неща, освен отговора (например тестови или междинни резултати), решението Ви да не бъде зачетено като коректно и да не получите точки за него.

## Cheat sheet

Извеждане на float число със закръгление до втората цифра след десетичния знак:

print("{:.2f}".format(2.456526))

2.46

<3Python!