1.**SOLID** principles:

* Single-responsibility principle (SRP): всеки клас трябва да е отговорен само за една задача и това трябва да се осъществява с методите му. Ако един клас се грижи за повече от една задача, то трябва да ги разделим в различни класове.
* Open–closed principle (OCP): класовете трябва да могат да се екстрендват, но да не могат да се модифицират.
* Liskov substitution principle (LSP): Принципът казва, че обектите на суперкласа трябва да могат да бъдет заместени с обекти от суб-класа, без това да счупи приложението. Това означава, че обектите от суб-класа трябва да имат еднакво поведение с това на обектите от суперкласа.
* Interface segregation principle (ISP): Класовете не трябва да са обвързани с методи, които не използват. Ако един клас не използва дадените му методи, те трябва да са отделени в по-специфични класове.
* Dependency inversion principle (DIP): абстракциите не трябва да зависят от детайли, а обратното. Един клас не трябва да зависи от друг клас, и за това можем да създадем междинен абстрактен метод

2. **Encapsulation**: Един от основните принципи на ООП. Той слага ограничения на това от къде може да се достъпва дадена променлива или метод, и може да предотврати модифициране на данни. За това, променливите могат да бъдат променени само чрез метод. Тези променливи се наричат private.

3.**Authorization**: what you can do? Има различни нива на достъп – superuser, user и може да има и групи, които да са оторизирани да достъпват еднаква информация.

4.**Authenticaion**: who you are? Използват се креденшъли(сингъл и мулти-фактор)

5. **venv**:  позволява ни да инсталираме различни пакети за различните ни проекти. Позволява ни да създадем виртуална изолирана среда, и там да инсталираме пакети.

6. **HTTP and HTTPs**: (HyperText Transfer Protocol) HTTP e стандарт за комуникация между уеб браузъра и уеб сършвъра, за да се установи комуникация. То е сет от правила за трансфериране на данни от 1 комп на др. HTTPS е по-сигурно от HTTP, защото има SSL сертификат.

7.**PostgreSQL**: open-source релационна ДБ. Поддържа SQL (релационни) and JSON (нерелационни) querying.

8. **Getters** and **Setters**: Гетърите и сетърите основно се използват за енкапсулиране на данни. Private vairabli-те в Пайтън не са в действителност скрити полета като в друите обектно ориентирани езици. Използваме ги когато:

* Искаме да добавим логика за валидиране на дадена стойност
* Да избегнем директен достъп до дадено поле в нашия клас. Private променливите не могат да бъдат достъпени или модифицирани директно.

9.**Code protection** – how to do it with getters and setters: Чрез гетъри и сетъри моцем да направим прайвът дадени променливи от нашия клас, така че те да не могат да бъдат достъпвани или модифицирани директно. Това можем да го постигнем с @property декоратора

10.**Relational vs Non-Relational DB/SQL and NoSQL/**: релационните ДБ съхраняват данни в таблици с редове и колони, където вс ред редпрезентира запис, а вс колона – поле(MariaDB, Postgre, MySQL). Нерелационни ДБ не използват традиционната таблична структура – използват key-value стойности(Redis, MongoDB)

11.**Python**: динамичен, high-level, open source, програмен език. Поддържа ООП.

12.**Unit vs Integration Testing**: unit testing-a се прави върху един малка изолирана част от кода, като например само 1 функция, докато integration testing-a се прилага върху цял flow на апликейшъна.

13.**Tuple:** Tuple-ите са подредени(всеки елемент има конктретно място. Веднъж създаден, не можем да добавяме или махаме елементи. Може да има елементи с еднакви стойности. Елементите имат индекси, като първият е с индекс 0.

14. **List**: Елементите в list-овете са подредени(ако добавим ел – отива накрая на листа), можем да добавяме и махаме елементи, след като е създаден, и може да съдържа елементи, които са еднакви. Елементите са индексирани, като първият е под индекс 0.

15.**‘Self’**: „Селф“ параметърът е референция към текущата инстанция на класа и се използва за достъпване на променливи към класа. Не е нужно да се казва „селф“, но така е прието, и трябва да бъде първият параметър от всички функции към класа.

16. **Module** vs **Library** vs **Package**:

* **Module**: съвкупност от свързан код, който се запазва във файл с разширение .py. Може да изберем да дефинираме функции, класове или променливи във модулите.
* **Library**: Бибилиотеката представлява преизползваем код. Обикновено съдържат модули и пакети.
* **Package**: пакетите са директория, която съдържа различни модули. Те ни позволяват да имаме йерархия и да направим кода си по-структуриран. Всеки пакет трябва да има ––инит.пу–– файл

17. **RegEx**: (регулярни изрази) поредица от символи, които сформират search pattern. Например, могат да се използват, за да се провери дали стринг съдържа определен search pattern.

18. **Lambda**: анонимни функции, което означава, че не са наименовани. Могат да имат няколко аргумента, но се връща само 1 нещо.

| **[List]** | **(Tuple)** | **{Set}** | **{Dictionary}** |
| --- | --- | --- | --- |
| A list is a non-homogeneous data structure that stores the elements in columns of a single row or multiple rows. | A Tuple is also a non-homogeneous data structure that stores elements in columns of a single row or multiple rows. | The set data structure is also a non-homogeneous data structure but stores the elements in a single row. | A dictionary is also a non-homogeneous data structure that stores key-value pairs. |
| The list allows duplicate elements | Tuple allows duplicate elements | The Set will not allow duplicate elements | The dictionary doesn’t allow duplicate keys. |
| Example: [1, 2, 3, 4, 5] | Example: (1, 2, 3, 4, 5) | Example: {1, 2, 3, 4, 5} | Example: {1: “a”, 2: “b”, 3: “c”, 4: “d”, 5: “e”} |
| A list is mutable i.e we can make any changes in the list. | A tuple is immutable i.e we can not make any changes in the tuple. | A set is mutable i.e we can make any changes in the set, ut elements are not duplicated. | A dictionary is mutable, ut Keys are not duplicated. |
| List is ordered | Tuple is ordered | Set is unordered | Dictionary is ordered |