

# Notes

...

Telegram: @shahin\_lri

## Headlines

### 0) Basics for beginners

- 1) Types of programming languages
- 2) Static, dynamic, strong and weak typing languages
- 3) Programming paradigms

### 1) Installation and implementation

- 1) Pycharm

### 2) Basics and rules of syntax

- 1) Syntax, physical lines, logical lines
- 2) Variable
- 3) Operators
- 4) Expression and statement

### 3) Python data types

- 1) Numeric
- 2) Dictionary
- 3) Boolean
- 4) Set
- 5) Sequence Type
  - 1) String
    - 1) Encryption and Decryption

- 2) Encoding and Decoding
- 3) Bytes data type
- 2) List
  - 1) Type of copy
- 3) Tuple
- 6) Mutable – Immutable
- 4) Control commands(decision)
- 5) Control commands(repeat)
  - 1) Break-continue-else in loop
  - 2) Loop technique
- 6) Functions
  - 1) Function name
  - 2) Parameter and argument
  - 3) Documentation string
  - 4) First class
  - 5) Name space – scope
  - 6) Pass by value – pass by reference
  - 7) Lambda
  - 8) Iterator
  - 9) Generator
  - 10) Coroutine
  - 11) Decorator
    - 1) decorator in function
    - 2) Decorator in classes
  - 12) Function attributes
  - 13) Recursive function
    - 1) Decorator and recursive generator function – recursive depth
    - 2) Memoization technique

## 7) Comprehension

- 1) List Comprehension
- 2) Generator Expression
- 3) Set Comprehension
- 4) Dictionary comprehension

## 8) Modular programming

- 1) Concepts of scripts, modules, packages, libraries, frameworks
- 2) Building and use the Module and Package
- 3) Environment variables and Python versions
- 4) Package manager
- 5) Virtual environment

## 9) File

- 1) Open function
- 2) Context manager 12-16-26
- 3) Shey file standard
- 4) JSON
- 5) CSV

## 10) Class HEADLINE

- 1) oop
  - 1) Encapsulation
  - 2) Abstraction
    - 1) Abstract class 12
    - 2) The type of relationship between objects 12
  - 3) Inheritance
    - 1) Mixin
  - 4) Polymorphism
    - 1) Duck typing
  - 2) Rules in oop
  - 3) concepts in oop

4) <u>property</u>	10
5) <u>Descriptor</u>	12-16-26
6) <u>Callable objects</u>	
7) <u>Type of methods and attributes</u>	
8) <u>object initialization(object's methods)</u>	
1) <u>Limit attributes</u>	29-7-17
9) <u>Metaclass</u>	26-2-12
10) <u>Dataclass</u>	12-16-26
11) <u>Error management</u>	
12) <u>Built-in functions</u>	
13) <u>Modules</u>	
1) <u>random</u>	
2) <u>os</u>	
3) <u>importlib</u>	
4) <u>re</u>	
5) <u>Logging</u>	
6) <u>other</u>	
14) <u>Packages</u>	

## Rules

سر فصل ها با رنگ قرمز در وسط با کاراکتر شروع بزرگ

سر موضوع ها با رنگ آبی در وسط کاراکتر شروع بزرگ

هر بخش موضوع با رنگ آبی در ابتدای خط کاراکتر شروع بزرگ

توضیح و زیر بخش برای متن بالایی با رنگ متن سبز اول و به ترتیب

خود note با رنگ قرمز و توضیحش با رنگ آبی.

نتیجه‌گیری یک موضوع با رنگ سبز.

مقدمه درباره مبحث جاری با رنگ آبی.

خلاصه‌ی فصل با رنگ آبی در وسط.

کد هر مبحث با Cx

توضیح هر تکه کد با Nx

پیش‌نیاز در ابتدای موضوع با سین تکس زیر می‌باشد

prerequisites: [iterator](#).

مثال‌ها

Example for recursive function:

(ex1) برنامه محاسبه فاکتوریل: ...

اگه کد پاین یک مبحث در خط جدا باشه یعنی مربوط به کل توضیحات خط ها بالایی هست

pych - #60 : این کد در پای چارم تست بشه  
syn #62: سین تکس یک کد و اجرا نداره

دیتا تایپ

رنگ متدها سبز و زیرن بنفش

جدول ها با سایز ۱۵

متن سلولها با لینک #000080 و بدون لینک سبز

## Walrus protocols:

• توصیه شده که در هر جا که نیاز بود از عملگر والروس استفاده شود داخل پرانتز تعریف شود.

: بجث در رابطه با مسله جاری با رنگ سبز import moduleName as mn

Red means coercion, like coercion input and green is optional

## 0) Basics for beginners

### Types of programming languages

ویژگی زبان های سطح بالا: قابلیت انتزاع(مفاهیم کلی مثل حیوان، ماشین ...) دارند، به زبان انسان نزدیکتر هستند، خوانایی بهتری دارند، مدیریت حافظه و مدیریت مستقیم پردازند را خودشان انجام میدهند(سی شارپ، روبی، پایتون، جاوا و جاوا اسکریپت).

مزایا : یادگیری ساده تر، مناسب برای وب و گیم، امنیت بالاتر.

معایب : سرعت کمتر، کنترل سخت تر حافظه و سخت افزار، نامناسب برای نوشتن کرنل، ویندوز.

ویژگی زبان های سطح میانی: برخی از قابلیت های هر دو سطح بالا و پایین را دارند مثلاً قابلیت مدیریت حافظه و قابلیت انتزاع (سی، سی پلاس پلاس).

ویژگی زبان های سطح پایین: قابلیت انتزاع ندارند، توسط ماشین خوانده می شوند و به زبان انسان نزدیک نیستند، نیاز به مدیریت حافظه دارند (زبان ماشین(0,1)، اسembلی).

مزایا : سرعت بیشتر، کنترل حافظه و سخت افزار، مناسب برای نوشتن کرنل، ویندوز.

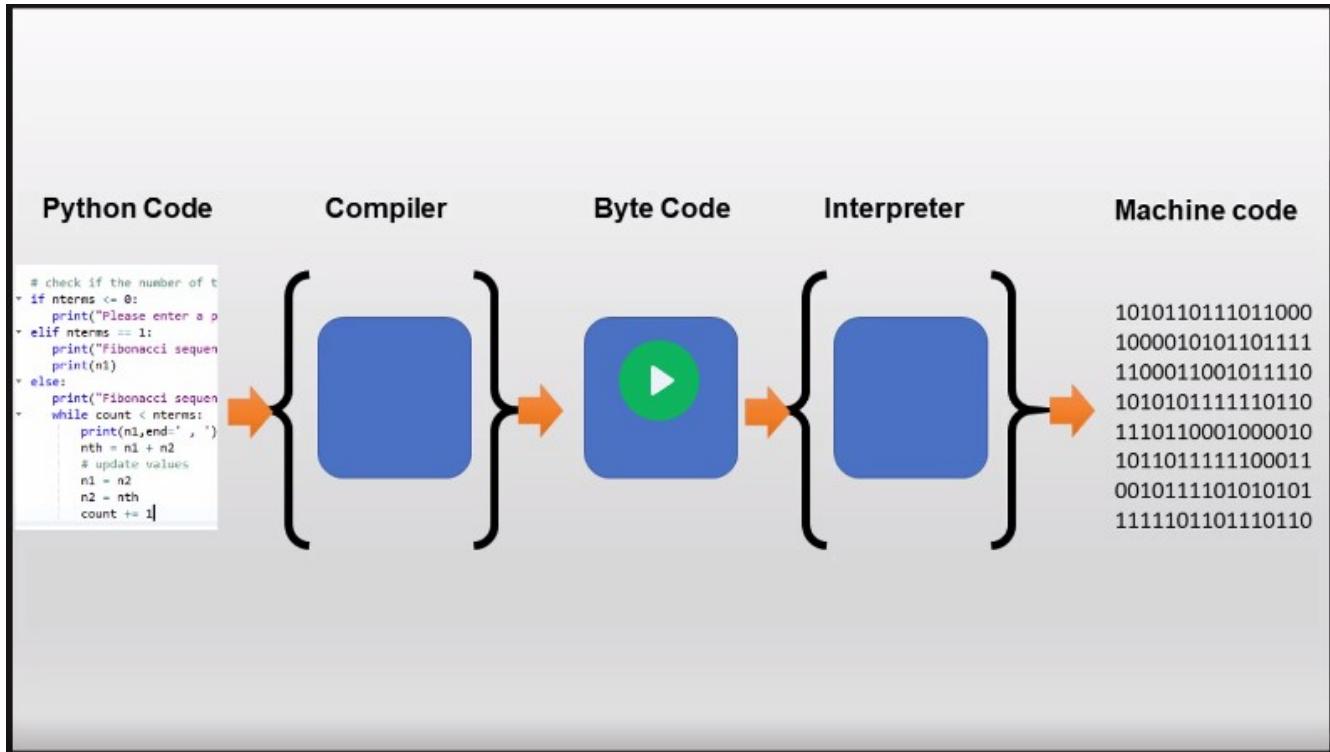
معایب: یادگیری سخت تر، نامناسب برای وب و گیم، امنیت کمتر.

زبان های همه منظوره/General Purpose Language: زبان هایی هستند که در چند حوزه و فیلد مختلف استفاده می شوند(python, java).

زبان های خاص منظوره/Domain Specific Languages: زبان هایی هستند که فقط برای یک هدف خاص طراحی شده اند(.html,css, sql, xml).

برنامه به یک فایل تبدیل می شود و سپس اجرا می شود، زبان های کامپایلری وابسته به سیستم عاملند، سرعت بیشتری دارند، استفاده کمتر از حافظه و پردازند، خطایابی سخت(C,C++, C#, pascal, cobol and so on).

زبان‌ها مفسری خط به خط اجرا می‌شوند، به سیستم عامل وابسته نیستند، سرعت کمتر، خطایابی آسان(Interpreter.(perl, php, python, ruby, javascript)



## Static, dynamic, strong and weak typing languages

چک کردن نوع داده در زبان‌های استاتیک تایپ در قبل از اجرای برنامه و در زبان‌های داینامیک تایپ در زمان اجرای برنامه انجام می‌شود برای بررسی اینکه از قوانین نوع پیروی می‌شود یا نه.

به زبان‌هایی که قبل از تعریف نوع، نوع متغیر را مشخص می‌کنیم زبان‌های تایپ استاتیک گفته می‌شود (static type). (c, c++, c#, java, kotlin).

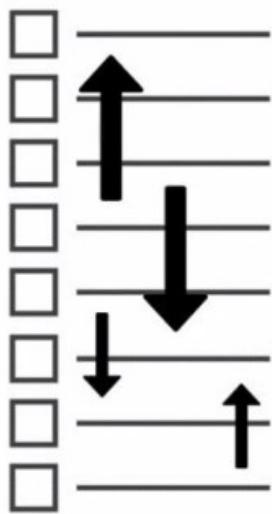
زبان‌هایی که قبل از تعریف داده نیاز نیست نوع آن را مشخص کنیم و نوع آن بصورت خودکار مشخص می‌شود (dynamic type) (python, php, javascript, ruby, perl).

معمولًا زبان‌های کامپایلری استاتیک تایپ هستند و زبان‌های مفسری داینامیک تایپ.

در این نوع زبان‌ها حساسیت روی نوع داده‌ها زیاده مثلاً اجازه نمیده یک عدد رو با یک رشته جمع کنیم (strong type).

در این نوع زبان‌ها حساسیت روی نوع داده‌ها کم مثلاً اجازه میده یک عدد رو با یک رشته جمع کنیم (weak type). (c, php, javascript).

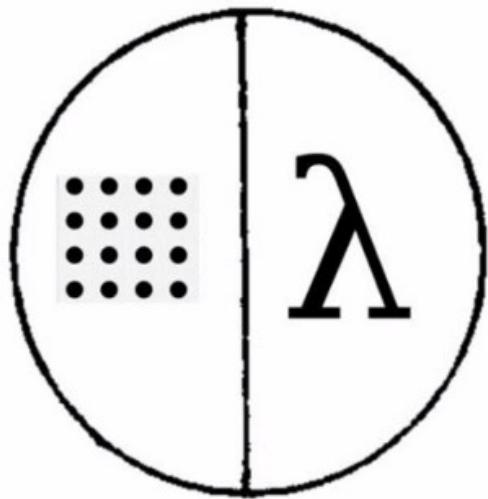
Imperative

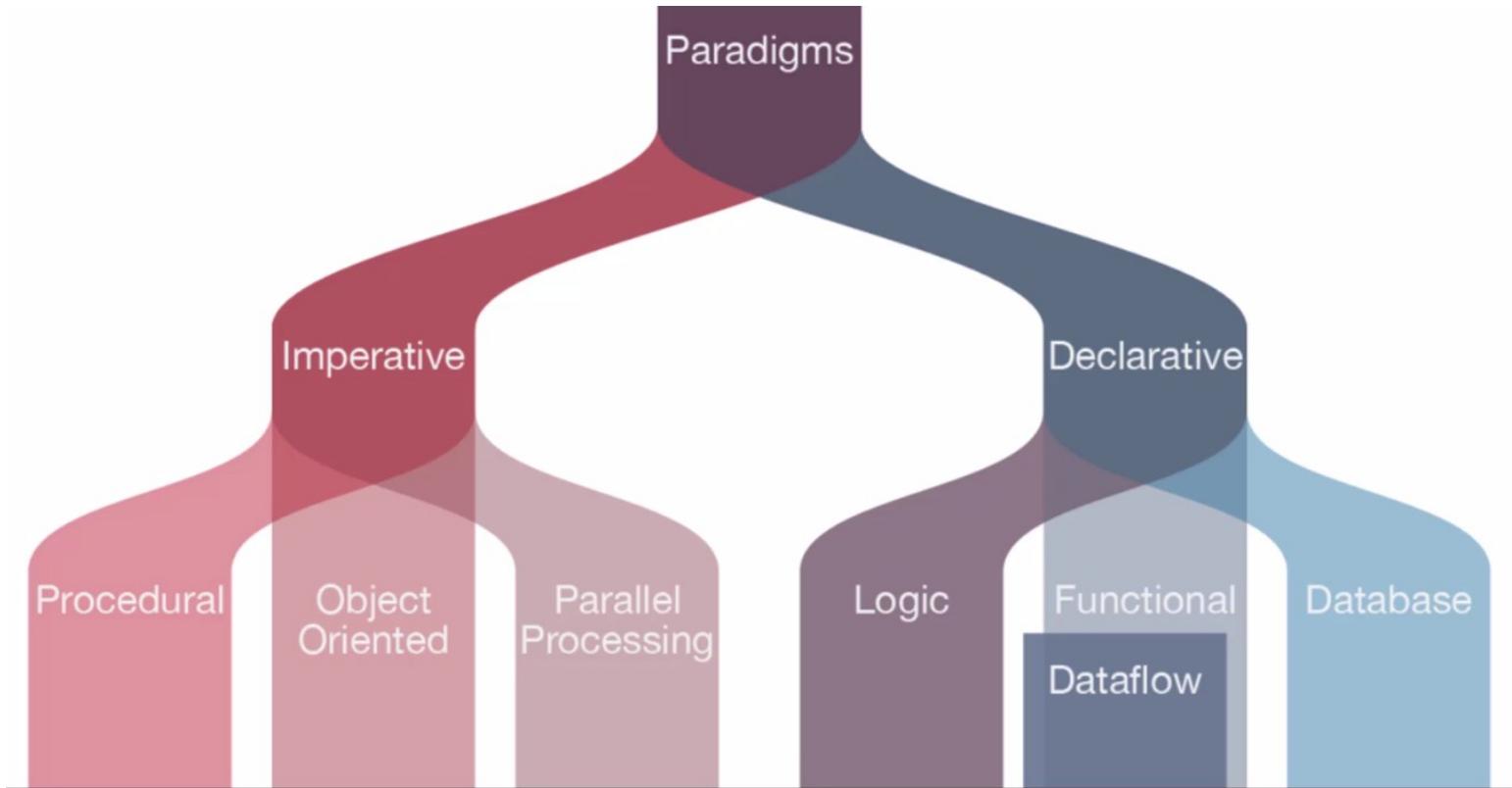


Functional



Object-Oriented





- Imperative

- Procedural:

هر مرحله باید بصورت قدم به قدم مشخص بشه .(c,c++, java, pascal)

- Object Oriented:

این روش از دنیای واقعی الهام گرفته شده

.(c#, c++, objective c, java, python, ruby, visual basic)

- Parallel Processing:

برای سرعت بیشتری کارهارو روی پردازنده ها تقسیم میکنه تا  
پردازش روی چند پردازنده انجام شود و درنهایت با ترکیب نتیجه  
پردازش های پردازنده های مختلف نتیجه کار رو میده.

- Declarative

- Logical:

بر روی منطق ریاضی هست و براساس دیتای موجود و فرضیات به

نتیجه دست پیدا میکنه(آدم‌ها میمیرند، رضا هم آدم هست، پس رضا هم خواهد مرد)(prolog).

- Functional:

کل برنامه به تابع‌های مختلف تقسیم میشے.

- Database:

فقط با دیتا سرو کار داره(sql).

## 1) Installation and implementation

### Pycharm

IDLE = Integrated Development Environment

shortcuts:

- view/Quick Definition - for guide of code
- view/Quick Documentation - for guide of code
- ctrl+b - for view docstrings
- ctrl+~ - quick access to pycharm appearance settings
- ctrl+shift+a - search between pycharm settings.
- shift+enter - new line

## 2) Basics and rules of syntax

### Syntax, physical lines, logical lines

: قواعد و قوانین نحوه نوشتن کد در هر زبان که نسبت به هر زبانی متفاوت هست. syntax

: سطرهایی که به ظاهر یک سطر هستند و معمولاً شماره گذاری هم دارند. physical lines

: سطرهایی که از نظر مفسر یک سطر هستند. logical lines

### Variable

: سی پپ حتی برای مقدارهای تکراری هم یک مکان جدید از حافظه را اختصاص میدهد. Variables in cpp

: در پایتون بخلاف سی پپ که برای هر مقدار یک خانه‌ی جدیدی از حافظه را اختصاص میدهد، پایتون یک لیست از اسمی متغیرها دارد و اسمی را به خانه‌های حافظه ارجاع (reference) میدهد و اگر هرچه قدر هم متغیر با مقادیر تکراری تعریف شود برای هر مقدار تکراری یک مکان جدید از حافظه اختصاص نمیدهد و هر متغیر با مقدار تکراری را به یک مکان حافظه که از قبل با همان مقدار تعریف شده ارجاع میده.

: در مواقعي که یک مقداری در حافظه قبل ایجاد شده و در حال حاضر هیچ ارجاعی بهش نشده معمولاً اون داده اگر مقدار کوچیک باشه نگه داشته میشه تا در صورت تعریف مجدد همون مقدار ازش استفاده بشه ولی اگر اون داده بزرگ باشه از بین میبره. Note

## Operators

### Arithmetic Operators:

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	$x / y$
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

## Assignment Operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x // 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x  = 3	x = x   3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

## Comparison Operators:

Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>&gt;</code>	Greater than	<code>x &gt; y</code>
<code>&lt;</code>	Less than	<code>x &lt; y</code>
<code>&gt;=</code>	Greater than or equal to	<code>x &gt;= y</code>
<code>&lt;=</code>	Less than or equal to	<code>x &lt;= y</code>

## Logical Operators:

Operator	Description	Example
and	Returns True if both statements are true	$x < 5 \text{ and } x < 10$
or	Returns True if one of the statements is true	$x < 5 \text{ or } x < 4$
not	Reverse the result, returns False if the result is true	<code>not(x &lt; 5 and x &lt; 10)</code>

Logical operators with data type: C15.

## Identity Operators:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

## Membership Operators:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

## Bitwise Operators:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	$x \& y$
	OR	Sets each bit to 1 if one of two bits is 1	$x   y$
^	XOR	Sets each bit to 1 if only one of two bits is 1	$x ^ y$
~	NOT	Inverts all the bits	$\sim x$
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	$x << 2$
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	$x >> 2$

Walrus operator:



عملگر والروس عملگری هست که به کمک اون همزمان هم مقدار دهی میکنیم و هم میتوانیم به اون مقدار دسترسی داشته باشیم.  
و عمدۀ کاربردش تو شرط، حلقه، لیست، و ... هست.

Walrus protocols:

- توصیه شده که در هر جا که نیاز بود از عملگر والروس استفاده شود داخل پرانتز تعریف شود.

روش‌های استفاده از عملگر والروس:  
(ex1) سین تکس استفاده از عملگر والروس در حلقه: دریافت و مقایسه و ذخیره یک مقدار در لیست .C79

(ex2) سین تکس استفاده از عملگر والروس در شرط .C62

- .C63) سین تکس های استفاده از عملگر والروس و با function annotation و ... در تابع (ex3
- .C64) سین تکس استفاده از عملگر والروس در اف استرینگ (ex4
- .C65) سین تکس استفاده از عملگر والروس در دیکشنری (ex5

## Python Operators Precedence:

Level	Operator	Description
18	()	Grouping
17	f()	Function call
16	[index:index]	Slicing
15	[]	Array Subscription
14	**	Exponential
13	~	Bitwise NOT
12	+ -	Unary plus / minus
11	*	Multiplication
	/	Division
	%	Modulo
10	+ -	Addition / Subtraction
9	<<	Bitwise Left Shift
	>>	Bitwise Right Shift
8	&	Bitwise AND
7	^	Bitwise XOR
6		Bitwise OR
5	in , not in, is , is not	Membership
	<, <=, >, >=	Relational
	==	Equality
	!=	Inequality
4	not	Boolean NOT
3	and	Boolean AND
2	or	Boolean OR
1	lambda	Lambda Expression

Operator	Description
( )	Parentheses
**	Exponentiation
+X -X ~X	Unary plus, unary minus, and bitwise NOT
* / // %	Multiplication, division, floor division, and modulus
+ -	Addition and subtraction
<< >>	Bitwise left and right shifts
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
== != > >= < <= is is not in not in	Comparisons, identity, and membership operators
not	Logical NOT
and	AND
or	OR

## Expression and statement

: عبارت حداقل یک مقداری تولید میکند و باید ارزیابی بشه و معمولاً عبارت رو میتوانیم بدیم به متغیر یا پرینتش کنیم و عبارت‌ها شامل شناسه، لیترال، عملگرهای حسابی میشون.

Ex:  $x+7*8$

: دستور متشکل از عبارت‌ها و کلمات کلیدی و غیره هست و یه کاری یا یه دستوری رو انجام میده.

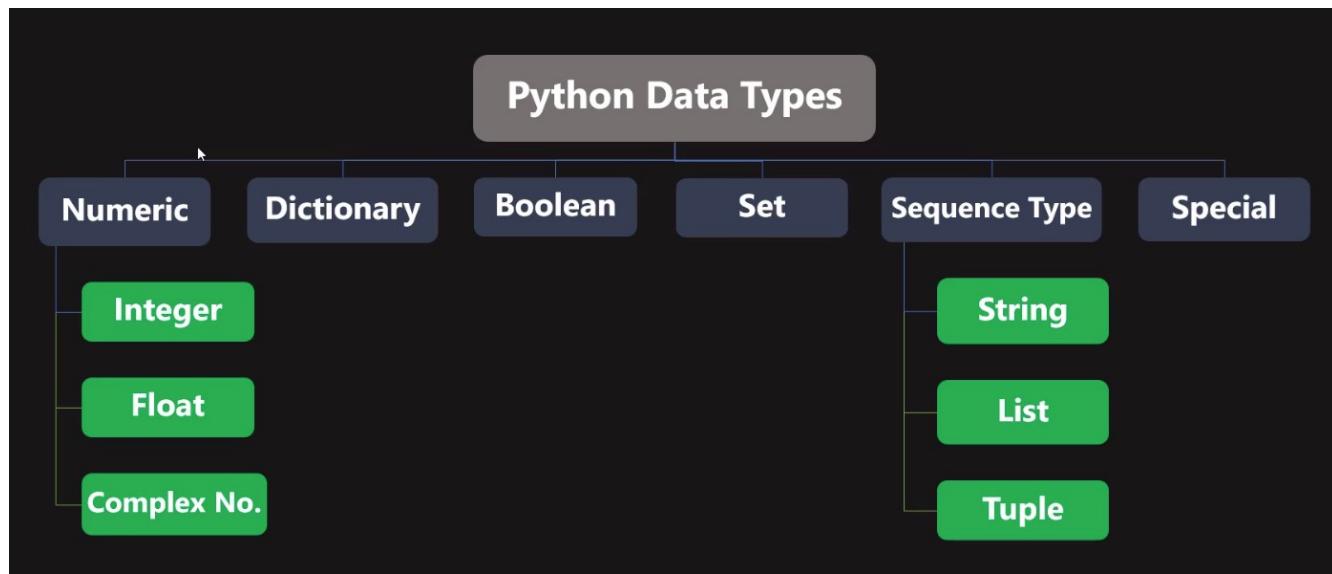
: دستوراتی هستند که در یک سطر منطقی پیاده میشوند. (A)

: معمولاً با یک کلمه کلیدی شروع و با کلون(:) تمام میشن که اصطلاحاً به آن بخش سرآیند گفته میشه و بعد در ادامه بدنشون نوشته میشه و این دستورات به دو بخش زیر تقسیم بندی میشون.

(A) **a-part / یک بخش:** مثل تابع.

(B) **conditional statement / multi-part:** مثل

### 3) Python data types(level one)



#### Numeric

- A) Integer
- B) Float
- C) Complex No.

#### Number functions:

rond(x, y)	مقدار x رو تا y رقم اعشار گرد میکنه.
abs	قدر مطلق
pow(x, y, mod)	اولی به توان دومی سپس نتیجه باقیماندش به mod حساب و برمیگردونه.
divmod(x, y)	حاصل y//x و y%x را داخل یک تابل برمیگردونه(از اسمش هم پیداست).

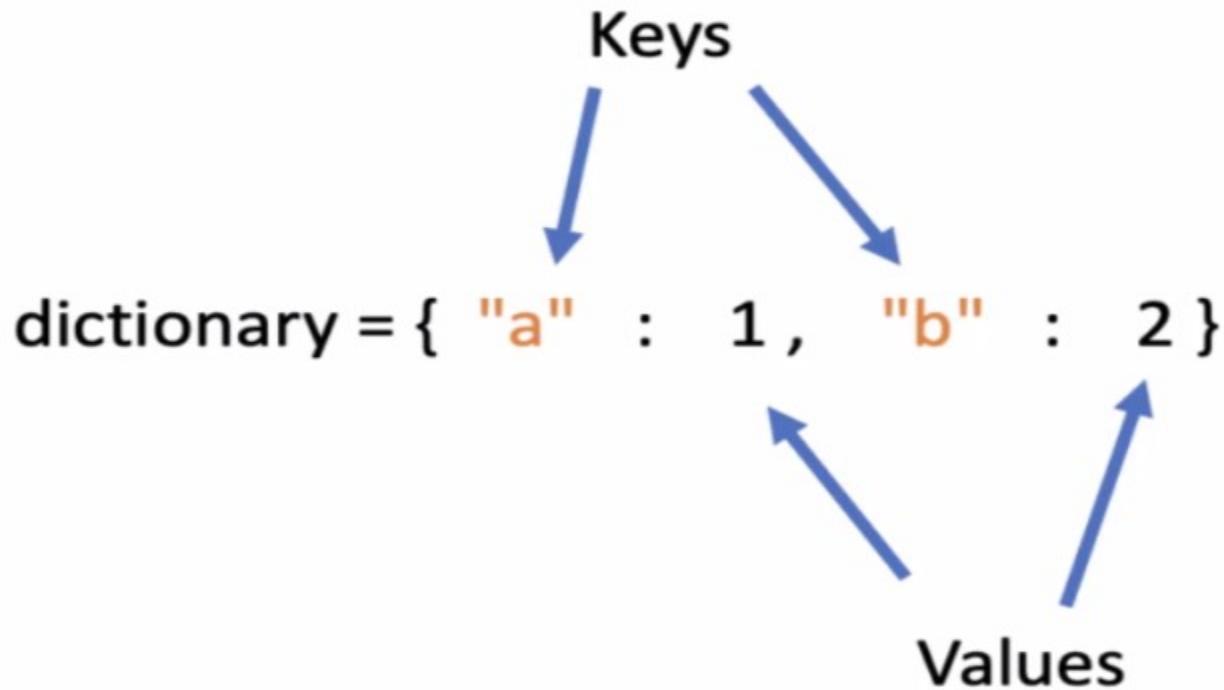
## Number methods:

<code>as_integer_ratio</code>	نسبت رو پیدا میکنه (مثلا نسبت 0.75 میشه 3.4).
<code>bit_count</code>	مثلاً باینری عدد ده 1010 این متده تعداد یک هارو میشمره.
<code>Real</code>	قسمت حقیقی عددرو نشون میده.
<code>Imag</code>	قسمت موهومی عددرو نشون میده.
<code>Conjugate</code>	مزدوج عددرو نشون میده.
<code>(45).to_bytes(x, byteorder='big', signed=False)</code>	عدد رو به نوع داده bytes تبدیل میکنه و سپس اگه چاپ کنیم اگر ممکن بود بجای عدد کاراکترش رو چاپ میکنه
<code>X</code>	چند تا بایت نشون بده.
<code>Byteorder</code>	اگر big باشه بالارزش ترین بیت در ابتدای ارایه قرار میگیره و اگر little برعکس.
<code>Signed</code>	متم 2 باشه یا نه.
<code>from_bytes(b'\x00\x00\x0f', byteorder='big', signed=False)</code>	آرایه‌ای از byte هارو رو به عدد تبدیل میکنه.
<code>b'\x00\x00\x0f':</code>	آرایه‌ای که قراره به عدد تبدیل بشه.
<code>fromhex()</code>	

## Numbers base functions:

- bin > number to Binary.
- oct > number to Octal.
- hex > number to Hexadecimal.
- 0b... > writing number in binary base.
- 0o... > writing number in octal base.
- 0x... > writing number in hexadecimal.

## Dictionary



### Dictionary functions:

<code>dict</code>	
-------------------	--

<code>d["a"]</code>	getting value of a key.
<code>d["c"] = 8</code>	if c was of before, updates it value, Otherwise, it adds c to the dictionary.
<code>del ["a"]</code>	delete a key and value.
<code>a   b</code>	جمع دیکشنری a و b

### Dictionary methods:

<u><a href="#">clear()</a></u>	تمام آیتم های دیکشنری را پاک می کند.
<u><a href="#">copy()</a></u>	Returns a shallow copy of the dictionary

<u>dict.fromkeys()</u>	یک دیکشنری با کلیدها و مقدار مشخص برمیگرداند
<u>get()</u>	مقدار آیتمی که کلید آن مشخص شده است را برمیگرداند.
<u>items()</u>	یک view object شامل لیستی از جفت کلید/مقدار ها که هر کدام در یک تاپل هستند را در خروجی برمیگرداند.
<u>keys()</u>	لیست کلید های دیکشنری را به صورت یک view object در خروجی برمیگرداند. این view object در صورتی که کلیدی به دیکشنری اضافه یا کم شود، به طور اتوماتیک آپدیت می شود.
<u>pop()</u>	آیتم مشخص شده را از دیکشنری حذف می کند و آن را در خروجی برمی گرداند
<u>popitem()</u>	آخرین آیتمی که به دیکشنری اضافه شده است را حذف می کند
<u>setdefault()</u>	مقدار کلید مشخص شده را در خروجی برمیگرداند. اگر کلید وجود نداشت، با استفاده از مقدار پیشفرض انتخاب شده، آن آیتم را به دیکشنری اضافه میکند.
<u>update(k:v, k:v, ...)</u>	به تعداد دلخواه آیتم جدید به دیکشنری اضافه می کند و اگر کلیدی از قبل در دیکشنری وجود داشته باشد مقدارش را آپدیت میکند.
<u>update(a=1, b=2, ...)</u>	یه روش دیگه
<u>values()</u>	لیست مقدار های دیکشنری را به صورت یک view object در خروجی برمیگرداند

Dictionary techniques:

update or change dictionary keys C9.

Dictionary protocols:

: نوع داده‌های immutable برای کلید های دیکشنری استفاده میشود. Note

## Boolean

**True:** Negative and positive numbers

**False:** A data type that does not have a value, none type, 0

Boolean functions:

bool	
------	--

Boolean methods:

## Set



مجموعه شامل اشیایی هست که ویژگی‌ها یا صفات مشترکی با هم دارند مثل مجموعه حیوانات و ... ولی در کل بسته به همبندی مجموعه داره که مثلاً یه جایی انسان و حیوان در یک مجموعه باشند ولی در جای دیگر ممکن است در مجموعه های جداگانه باشند.

### Set protocols:

اگر عناصر تکراری در بین عناصر ست وجود داشته باشند موقع اجرا تکراری ها حذف میشن.  
مجموعه یک نوع داده **mutable** است.

شما نمی توانید نوع داده **mutable** را در داخل مجموعه تعریف کنید و باید

بشه immutable.

مجموعه دارای نظم نیست و از نمایه سازی و برش پشتیبانی نمی کند.

How to definition sets: C11.

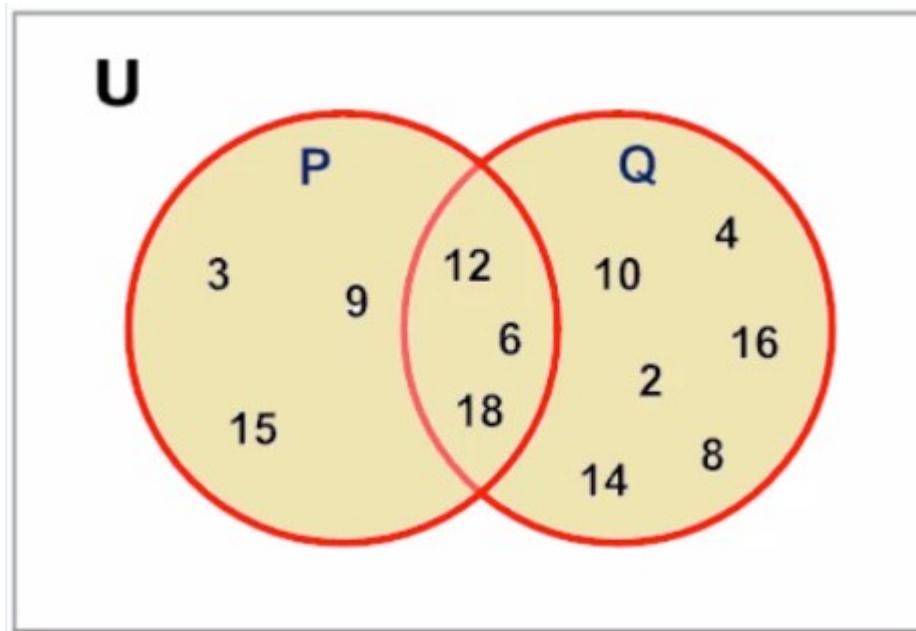
تفاضل: اونایی که در  $P$  هستند ولی در  $Q$  نیستند(جواب همیشه از اعضای اولین مجموعه است).

اجتماع: همه موارد موجود در  $P$  و  $Q$ .

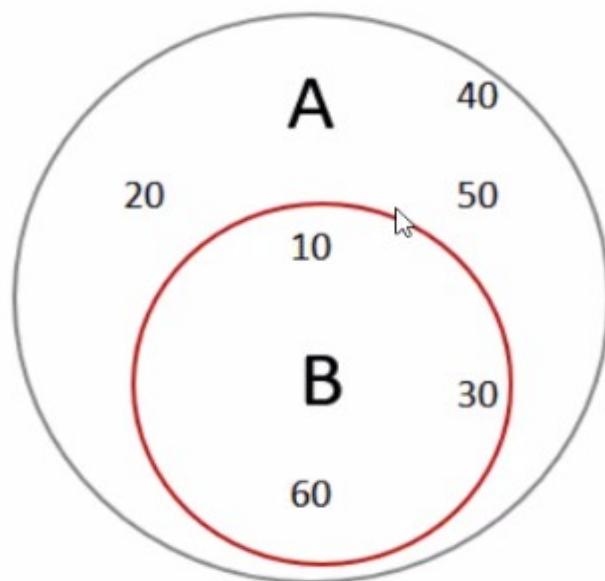
اشتراك: همه مواردی که هم در  $P$  و هم در  $Q$  هستند.

تفاضل متقارن: همه موارد، به غیر از اشتراك ها شون.

.C12



Subset and super set: C13.



## Set methods:

: متدهایی که آخر اسمش update وجود داره، یعنی هر عملی که توسط اون متدها انجام میشه **note** روی اون object هم اعمال میشه.

<u>add()</u>	Adds a given element to a set
<u>clear()</u>	Removes all elements from the set
<u>copy()</u>	Returns a shallow copy of the set
<u>difference()</u>	آیتم هایی از ست اصلی که در ست دوم وجود ندارد را در خروجی برمیگرداند
<u>difference-update()</u>	از ست اصلی آیتم هایی که با ست دیگر مشترک است را حذف می کند
<u>discard()</u>	Remove the specified item
<u>intersection()</u>	یک ست شامل آیتم های مشترک بین دو ست را در خروجی برمیگرداند
<u>intersection-update()</u>	آیتم هایی از ست که در ست های دیگر وجود نداشته باشد را از ست اصلی حذف می کند
<u>isdisjoint()</u>	عنصر مشترک در هر دو مجموعه باشه False برمیگردونه و نباشه True.
<u>issubset()</u>	اگر تمام آیتم های ست در یک ست دیگر وجود داشته باشد، True و در غیر اینصورت False برمی گرداند b is subset of a C13.
<u>issuperset()</u>	اگر تمام آیتم های ست دوم در ست اصلی باشد، True و در غیر اینصورت False برمیگرداند a is super set of b C13.
<u>pop()</u>	Returns and removes a random element from the set
<u>remove()</u>	Removes the specified element from the set
<u>symmetric-difference()</u>	یک ست شامل همه ی آیتم های دو ست به جز آیتم های مشترک را

	در خروجی برمیگرداند
<u><a href="#">union(s1,s2, ...)</a></u>	ادغام عناصر دو یا چند مجموعه برمیگرددونه.
<u><a href="#">symmetric-difference-update()</a></u>	آیتم های مشترک بین ست اصلی و ست دوم را از ست اصلی حذف می کند و آیتم های غیر مشترک را به آن اضافه می کند.
<u><a href="#">update()</a></u>	Adds elements to the set
<u><a href="#">frozenset()</a></u>	Return an immutable frozenset object

## Sequence Type

### A) String

string is immutable data type.

Indexing and Slicing:

x=HAMED

sequence	H	A	M	E	D
index+	0	1	2	3	4
index-	-5	-4	-3	-2	-1

Indexing

- $x[2]$  or  $x[-3]$  = M

Slicing

- $x[2:4:[stp]]$ =ME
- $x[1:3]$  or  $x[-4:-2]$  = AM
- $x[-2:]$  = ED
- $x[ : :-1]$  > sequence reverse

## String methods:

<u>capitalize()</u>	اولین حرف متن را به حرف بزرگ تبدیل می کند.
<u>casefold()</u>	تمام حروف متن را به حرف کوچک تبدیل می کند. قدرت بیشتری نسبت به lower دارد یعنی مثلًا شاید lower نتوانه یه سری کارکترهایی رو به کارکترهای کوچک تبدیل کنه این متده نمیتونه تبدیل بکنه مثلًا تو زبان آلمانی کارکتر آپرکسیس ؟ لاور کیسیش میشه ss که متده lower نمیتونه اینو تبدیل کنه).
<u>center(length, character)</u>	یک کلمه را در وسط قرار می دهد و دو طرف آن را تا رسیدن به یک طول n با یک کاراکتر مشخص (که به صورت پیش فرض فضای خالی است) پر می کند.
<u>count()</u>	تعداد باری که یک مقدار مشخص در یک متن استفاده شده است را در خروجی برمیگرداند
<u>encode()</u>	یک متن را بر اساس انکدینگ مشخص شده انکد میکند.
<u>endswith()</u>	اگر جمله یا کلمه ورودی با یک مقدار مشخص تمام شده باشد، True و در غیر اینصورت False بر می گرداند
<u>expandtabs(n)</u>	\t های متن رو با n تا space تعویض میکند.
<u>find()</u>	ایندکس اولین مکان پیدا شدن کلمه در متن را در خروجی برمیگرداند
<u>format()</u>	مقادیر ورودی را در جای مشخصشان در متن وارد می کند
<u>format_map()</u>	تقریباً شبیه format هست - هر کلید دیکشنری رو به کلید({x}) داخل رشته که با کلید دیکشنری همنام هستش سمت میکنه
<u>index()</u>	ایندکس اولین جایی که کلمه ی مشخص شده در متن پیدا می شود را در خروجی برمیگرداند. (تفاوتش با find اینه که در این اگه پیدا نکنه خطای میده ولی در find عدد 1- رو میده)

<a href="#">isdecimal()</a>	اعشاری هست؟
<a href="#">isdigit()</a>	رقم هست؟
<a href="#">isnumeric()</a>	عدد هست؟

isdecimal()	isdigit()	isnumeric()	Example
True	True	True	"038", "۰۳۸", "0 3 8"
False	True	True	"۰۳۸", "۰.۳۸", "۸۹۹"
False	False	True	"٪۱٪٪", "IIIIVIII", "۰۰۰", "壹貳參"
False	False	False	"abc", "38.0", "-38"

اعشار زیر مجموعه رقم هستش و رقم نیز زیر مجموعه عدد هست پس اگر اعشار true باشه هم رقم و هم عدد true هست. چیزی که اعشار میتونه باشه میتونه هم رقم و هم عدد باشه و ...

## More inf

<a href="#">isalpha()</a>	اگر تمام کاراکتر های متن جزو حروف الفبا باشند، True و در غیر اینصورت False برمیگرداند(مثلا space داشته باشه false میده)
<a href="#">isascii</a>	اگر تمام کاراکتر های متن جزو ascii باشند، True و در غیر اینصورت False برمیگرداند
<a href="#">isidentifier()</a>	اگر کلمه یک مشخصه‌ی معتبر (برای نام گذاری متغیرها و ...) باشد، True و در غیر اینصورت False برمی‌گرداند
<a href="#">islower()</a>	اگر تمام حروف متن، حرف کوچک باشد True و در غیر اینصورت False برمیگرداند.
<a href="#">isalnum()</a>	اگر تمام کاراکتر های متن، حرف الفبا(a-z) یا عدد(0-9) باشند، True و در غیر

	اینصورت False برمیگردد
<a href="#"><u>isprintable()</u></a>	اگر تمام کاراکتر های متن، قابل چاپ باشند، True و در غیر اینصورت False برمیگردد. (مثلا \t داشته باشه false میده، چون خود \t قابل چاپ نیست)
<a href="#"><u>isspace()</u></a>	اگر تمام کاراکتر های موجود در متن "فاصله خالی" باشند، True و در غیر این صورت، False برمیگردد.
<a href="#"><u>istitle()</u></a>	اگر تمام کلمات موجود در متن با حرف بزرگ شروع شوند و بقیه ی حروفشان، حرف کوچک باشد، True و در غیر اینصورت False برمی گردد.
<a href="#"><u>isupper()</u></a>	اگر تمام حروف متن به حرف بزرگ باشند، True و در غیر اینصورت False برمیگردد.
<a href="#"><u>join()</u></a>	تمام آیتم های آرایه ی ورودی را به شکل یک متن در کنار هم قرار می دهد
<a href="#"><u>ljust(n, "chr")</u></a>	کلمه را در سمت چپ قرار داده و ادامه ی آن را تا رسیدن به طول n با یک کاراکتر (که به صورت پیشفرض فضای خالی است) پر می کند.
<a href="#"><u>lower()</u></a>	تمام حروف متن را به حرف کوچک تبدیل میکند. اعداد و علائم در نظر گرفته نمی شوند
<a href="#"><u>lstrip()</u></a>	از سمت چپ متن کاراکتر های مشخص شده (که به صورت پیش فرض فضای خالی است) را حذف می کند
<a href="#"><u>partition()</u></a>	متن را بر اساس کلمه ی جستجو، به سه قسمت تقسیم کرده و به صورت یک تاپل در خروجی ارسال می کند
<a href="#"><u>rpartition()</u></a>	
<a href="#"><u>replace()</u></a>	کلمه مشخص شده را در متن با یک کلمه دیگر جایگزین میکند
<a href="#"><u>rfind()</u></a>	ایندکس اولین مکان پیدا شدن کلمه در متن را در خروجی برمیگردد.

<a href="#"><u>rindex()</u></a>	ایندکس آخرین جایی که کلمه در متن پیدا می شود را در خروجی برمیگرداند
<a href="#"><u>rjust(n, "chr")</u></a>	کلمه را در سمت راست قرار داده و ادامه‌ی آن را تا رسیدن به طول n با یک کاراکتر (که به صورت پیشفرض فضای خالی است) پر می‌کند
<a href="#"><u>rstrip()</u></a>	کاراکتر‌های مشخص (پیشفرض = فضای خالی) را از سمت راست متن حذف می‌کند
<a href="#"><u>split('-', 1)</u></a>	متن را به یک شکل مشخص تقسیم کرده و به یک لیست تبدیل می‌کند و براساس - جدا میکنه و 1 یعنی تا اولین - یک عنصر بشه و بقیه یک عنصر دیگه
<a href="#"><u>rsplit('-', 1)</u></a>	متن را از سمت راست بر اساس یک جدا کننده تقسیم می‌کند و یک لیست از آنها می‌سازد
<a href="#"><u>splitlines(True)</u></a>	تقسیم کردن یک متن و تبدیل آن به یک لیست به صورتی که هر خط از آن یک عضو لیست باشد - پارامتر اختیاری True به معنای اینه که separator رو نیز در عناصر نگه دار
<a href="#"><u>startswith()</u></a>	اگر متن با یک کلمه مشخص شروع شود، True و در غیر اینصورت False برمیگرداند
<a href="#"><u>strip()</u></a>	کاراکتر‌های مشخص شده (پیشفرض = فضای خالی) را از ابتدا و انتهای متن حذف می‌کند
<a href="#"><u>removeprefix('--')</u></a>	عین کارکتر یا کارکترهای ورودی را از ابتدا/انتها رشته حذف میکند -
<a href="#"><u>removesuffix('+')</u></a>	تفاوت: strip همه کارکترهای ورودیش را از اول یا انتهای رشته حذف میکند و ترتیب و تعداد براش مهم نیس ولی اینها فقط عین اوی که بهش به عنوان ورودی داده شده رو پاک میکنه.
<a href="#"><u>swapcase()</u></a>	تمام حروف بزرگ را به حروف کوچک تبدیل می‌کند و تمام حروف کوچک را به حرف

	بزرگ
<u>title()</u>	حرف اول تمام کلمات موجود در متن را به حرف بزرگ تبدیل کرده و بقیه را به حرف کوچک تبدیل می کند
<u>upper()</u>	تمام حروف متن را به حرف بزرگ تبدیل می کند
<u>zfill()</u>	به ابتدای متن صفر اضافه می کند تا آنجایی که به طول مشخص شده برسد

str.maketrans( <i>oldS, newS, delS</i> )	string.translate( <i>table</i> )
همان طور که از نام آنها پیداست یکی ساخت ترجمه و دیگری ترجمه هست، یعنی به کارکترهایی که میخوایم ترجمه میکنیم.	
دو متده هستند که معمولاً باهم استفاده میشوند.	
فرض کنید مثلاً میخوایم برخی کارکتر ها رو در یک متنی با کارکترهای دیگه جایگذین کنیم	
متده maketrans محتوای ورودی های خود را بصورت نظیر به نظیر encode میکند. یعنی اگر دو متغیر a,b رو در نظر بگیریم، کد کارکترهای متغیر a رو بعنوان key های dict قرار میدهد و کد کارکترهای متغیر b رو بعنوان value های dict قرار میدهد.	
مثلاً برای اولین کارکترهای نظیر به نظیر مقدار key، کد R هست و مقدار value کد r و ...	
حالا مثلاً میخوایم با استفاده از دیکشنری table که با متده maketrans ساختیم برخی از کارکترهای یک متن رو جایگزین کنیم.	
دیکشنری table رو به عنوان ورودی به translate میدیم تا بره در متن مورد نظر کارکترهارو جایگزین کنه، میره هر کارکتر رو تک تک بررسی میکنه و اگر کد هر کارکتر در text با کد هر کدام از key های dict(table) یکسان بود آن را که یک کد کارکتر هست تبدیل به کارکتر میکنه و بجاش replace میکنه .C77	

String formats: C2.

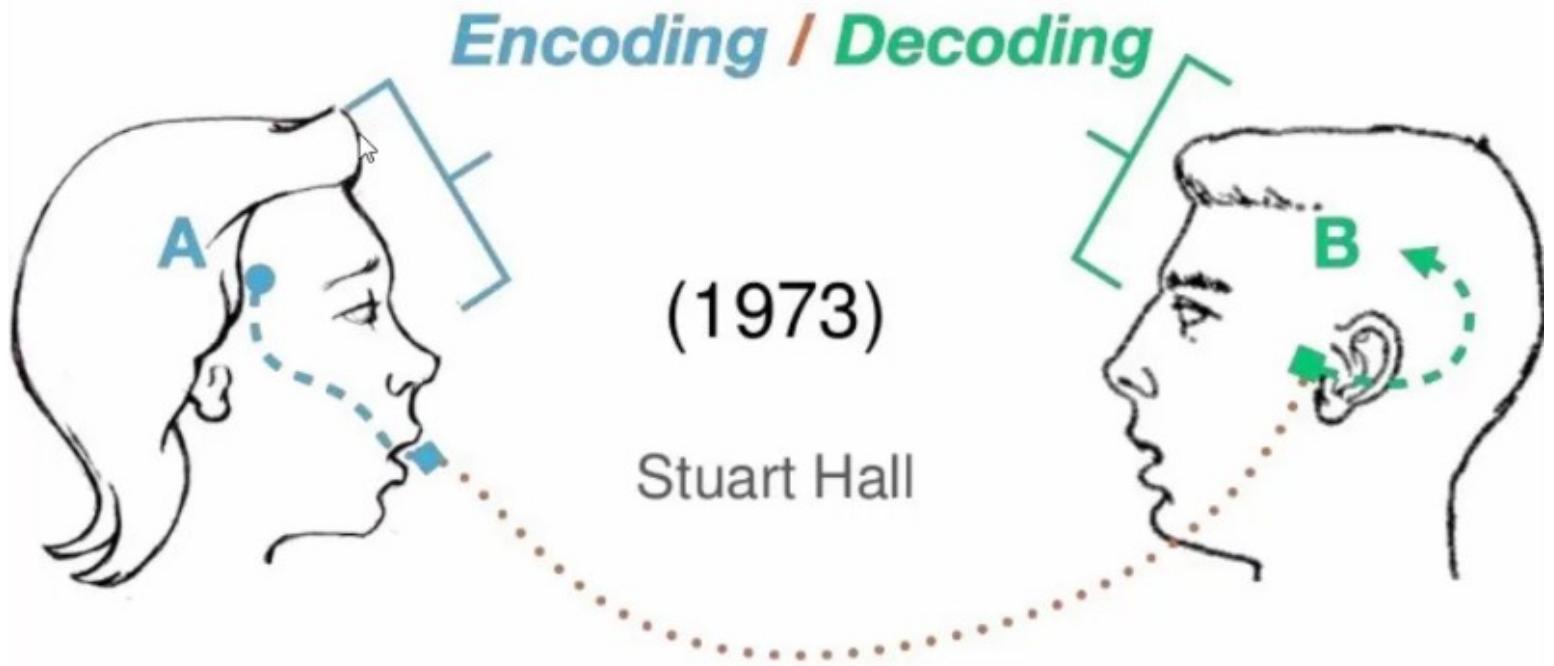
Backslash characters:

- \n > new line
- \b > backspace
- \t > tab
- \r > Back to the beginning of the current line
- \' > for print '
- \" > for print "
- \\ > for print \
- \u> for writing code unicode 32 bit
- \U> for writing code unicode 64 bit
- \ > for writing code unicode and code ascii from in octal base.
  - Ex: '\275'=%
- \x > for writing code unicode or code ascii from base hexadecimal.
  - Ex: '\x25' = %
- \' > cancels work of a statement
- \\ > cancels work of a slash
- r(raw string) > cancels work of a sequence(\n).
- """ \_ """ >

### How RSA Encryption Works



: برای حفظ امنیت داده‌ها اون داده‌ها طی الگوریتم هایی مخصوص رمزگذاری میشه تا افراد سودجو نتونن به اطلاعات ما دسترسی پیدا کنن.  
و فقط افرادی که کلید رمزگشایی دارن میتوونن به اطلاعات دسترسی پیدا کنن.



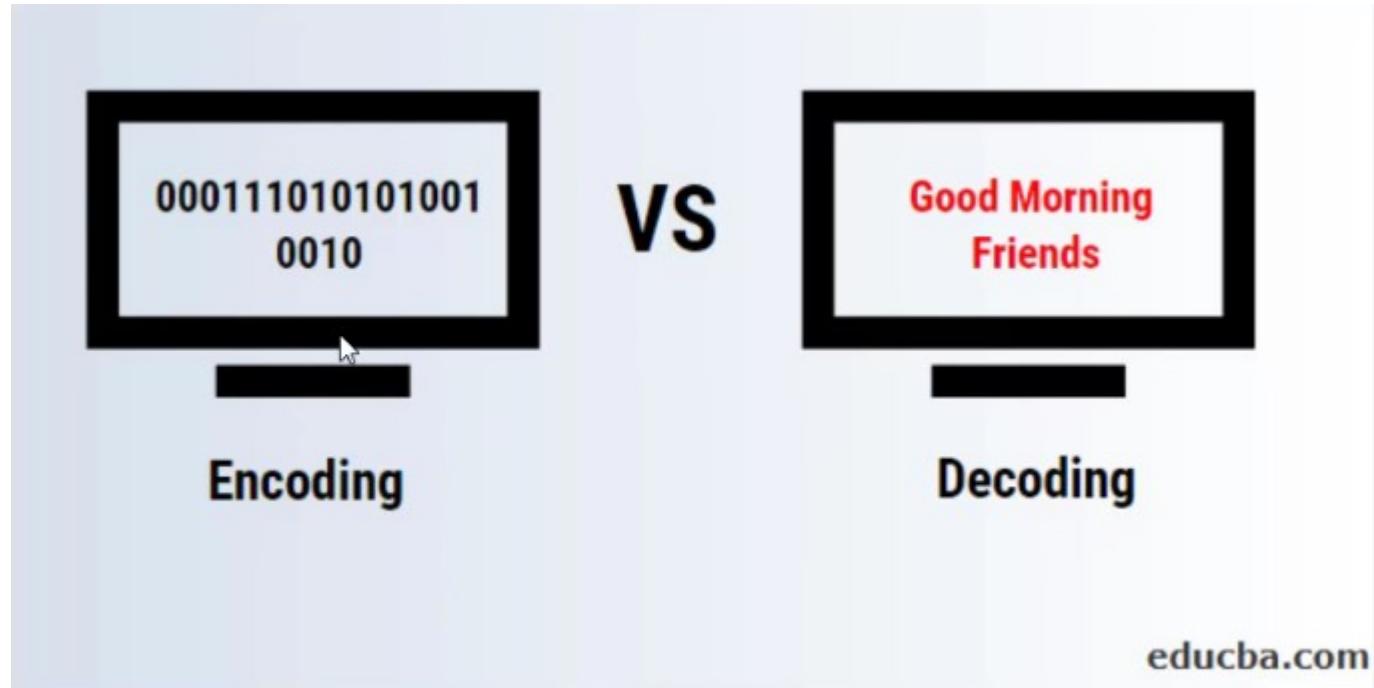
وقتی یه شخص میخواد یه حرف رو به یه شخص بزنه مجبوره که حرفهای خودش رو به زبان اون شخص ترجمه یا تبدیل یا معادل سازی کنه و اگه میخواد حرفهای اون شخص رو بفهمه باید حرفهای اون شخص رو به زبان خودش ترجمه کنه.

توی کامپیوتر نیز این پروسه وجود داره و برای اینکه حرفها یا دستورات ما رو کامپیوتر بفهمه باید اونو به زبان کامپیوتر تبدیل(کدگذاری) کنیم و اگه بخوایم زبان کامپیوترا رو به فهمیم باید اونو به زبان خودمون برگردانیم(کدگشایی کنیم).

مثلاً encoding یعنی جایگذین کردن کاراکترها و رشته‌هایی که برای کامپیوتر قابل فهم هست.

و decoding هم یعنی کاراکترها یا رشته‌هایی که برای کامپیوتر قابل فهمه و برای ما نیست به کاراکترها یا رشته‌هایی که برای ما قابل فهمه برگردانیم.  
یا تبدیل از زبان کامپیوتر به فرم اولیه.

Example:



کلمه‌ی Good Morning Friends همین‌جوری در حافظه ذخیره نمی‌شود بلکه اول طبق استاندارد مورد نظر encode می‌شود سپس به زبان ماشین(01) تبدیل می‌شود سپس در حافظه ذخیره و پردازش می‌شود.

$x = \text{Good Morning Friends}$

شرایط :encoding, decoding

- متنی که encode شده باید قابلیت decode داشته باشد.
- متن باید بدون decryption key decode قابلیت داشته باشد.

## [Encoding and decoding standards: more information.](#)

روش‌ها یا استاندارد هایی که با استفاده از اون به هر کاراکتری کد اختصاص داده (کد گذاری) می‌شه شامل موارد زیر می‌باشد.

- ASCII

اسکی برای هر کاراکتر از 8 بیت 7 بیت‌شن رو استفاده می‌کنه.  
از  $128^{(2^7)}$  کاراکتر پشتیبانی می‌کنه.

فقط از زبان انگلیسی و برخی کاراکترهای پر استفاده روی کیبورد پشتیبانی می‌کنه.

- **UNICODE**

از چند بایت می‌تونه برای کدگذاری استفاده کنه (یعنی از 8 یا 16 یا 32 بیت).  
از  $256^{(2^8)}$  کاراکتر پشتیبانی می‌کنه و کاراکترهای ascii یکسان می‌باشد.  
و از 128 به بعد در هر زبانی کاراکترها متفاوت می‌باشد.

- utf-8
- utf-16
- utf-32

### functions:

- ord > character to unicode or ascii.
- chr > unicode or ascii to character.

## Bytes data type

هر فایلی که تو پایتون ازش استفاده میکنیم، پایتون هر فایل رو byte date type در نظر میگیره، مثل: عکس، فیلم، متن، ...

توی پایتون 2 version، دو نوع رشته داشتیم:

:type str(normal) (a)

نوع داده‌ای بود که فقط از ascii و داده‌های binary(متن، عکس، فیلم ...) که encode شده اند پشتیبانی میکرد.

x = "n"	ascii
x = b"n"	binary اگر در content range اسکی باشه با b میشه به range نوع داده byte تبدیل کرد و اگه خارج اسکی باشه از متده استفاده میشه

:type unicode (b)

x = u"n"	unicode
----------	---------

اما در پایتون 3 کلاً ساختار رشته عوض شد و نوع های دیگه‌ای به وجود اومد:

:class str(normal)(a)

در پایتون ورژن سه str از ascii و unicode پشتیبانی میکنه. و نیاز نیست پشت literal کاراکتری نوشته بشه.

که class bytes(b) immutable هست.

وقتی داده‌ای رو با استفاده از کلاس bytes ذخیره میکنیم کد هر کاراکتر رو به مبنای باینریش تبدیل میکنه و سپس موقع چاپ اون کد باینری رو به کد hexadecimal(مبنای 16) اش

تبدیل و اگر بتوانه کارکترش رو از مبنای hexadecimal پرینت میکنه و اگر نتوانه همون کد print رو hexadecimal میکنه.

برخی کاراکترها وقتی به کد باینری تبدیل میشن بصورت 1 بایتی در میاند و برخی نیز بصورت 2 و 3 و 4.

:str (1)

مثلاً کاراکتری مثل کاراکتر `\n` هنگام تبدیل به کد باینری بصورت 2 بایتی ذخیره میشه و وقتی که بخوایم اش کنیم چون print Encoding and decoding standards تشخیص میده که این بصورت 2 بایتی ذخیره شده باخاطر اینکه نمیتونه این 2 بایت رو یکی کنه و کارکترش رو نشون بده پس کد decimal هر بایت رو بصورت جدا نمایش میده و بایت هایی که میتونه اونهارو یکی کنه رو، یکی میکنه و به کاراکترشون تبدیل میکنه و نمایش میده C73.

<code>x = b"n"</code>	It only supports ascii.
<code>bytes("test", "Encoding and decoding standards", "strict")</code>	Ascii, ... - Convert to bytes data type.
<code>strict</code>	default
<code>ignore</code>	Ignores the error if the input is not ascii.
<code>replace</code>	اونهایی که نمیشناسه رو با ? جایگزین میکنه
<code>backslashreplace</code>	اونهایی که نمیشناسه رو با \ جایگزین میکنه
<code>namereplace</code>	نوشتن نام کاراکتر بجای خود کاراکتر
<code>encode('utf-8')</code>	
<code>str(x, 'utf-8')</code>	Convert to string form bytes data type.
<code>x.decode('utf-8')</code>	

ولی نکته‌ای که وجود داره اینه که هنگام تبدیل به کد باینری بصورت چند بایتی درمیاند رو اگه هر بایت رو بصورت مجزا decode کنیم به یک کاراکتر دیگه دست پیدا میکنیم چون بصورت جدا جدا هم هر کدوم کد باینری یه کاراکتر خاصه ولی هر دو باهم هم بازم کد باینری یه کاراکتر خاصه.

مثلاً در **C75** کدهای `c3,a6` به ترتیب کد `hexadecimal` های کاراکترهای `ا, آ` هست ولی جفتش باهم کد `hexadecimal` کاراکتر `آ` هست.

**Note:** نوع داده `bytes` از `indexing` هم پشتیبانی میکنه. یعنی اگه بگیم ایندکس `x` رو چاپ کن، مقدار اون ایندکس رو به عدد معادلش در `decimal`(مبنای 10) تبدیل میکنه و چاپ میکنه **C78**.

`":Number (2`

داخل یک لیست اعداد بعنوان ورودی به متدهای `bytes` داده میشه که هر عنصر لیست یک بایت از حافظه رو اشغال میکنه که باید در رنج 0-255 باشه **C76**.

نحوه تشخیص اینکه آیا چند بایت مورد نظر در حافظه با هم مربوط به یک کاراکتر هست یا نه، در `:decode` هنگام

نحوه تشخیصش به این صورته که `Encoding and decoding standard` میفهمه از 2 بایت از اعداد باینری، اگر 2 رقم اول، اولین بایت 2 تا 1 بود این 2 بایتیه.

(**11**000010 10000000)

یا اگه از 3 بایت از اعداد باینری، اگر 3 رقم اول، اولین بایت 3 تا 1 بود این 3 بایتیه.

(**111**00010 10110110 10100000)

و اگه و ...

و بر اساس تعداد **1** تشخیص میده که چند بایت باهم تشکیل یک کاراکتر رو میدن

**C74**

هست mutable اے :class bytearray(C

"

bytearray("test", "Encoding and decoding standards")	Convert to bytearray data type.
--	---------------------------------

(d

## B) List

List is a mutable data type.

**note:** Like string, list supports indexing and slicing.

**note:** you can use the + and \* operator for adding two lists and \* for multiplication two lists.

How to define list: C4.

List techniques:

techniques	Description	method
copy   list[:]	get a shallow copy of list value.	
List=[:], del[:2]	clear all values of list.	clear
del list[2]	delete a element of a list.	
list[ : :-1]	Reverses the list.	reverse

List functions:

List methods:

<u>append()</u>	یک آیتم را به انتهای لیست اضافه می کند.
<u>clear()</u>	تمام آیتم های درون لیست را پاک می کند.
<u>copy()</u>	It returns a shallow copy of a list.
<u>count()</u>	تعداد باری که یک آیتم مشخص در لیست آمده است را برمیگردداند.
<u>extend()</u>	آیتم های یک لیست یا هر آرایه ی دیگر را به انتهای لیست اصلی اضافه می کند.
<u>index()</u>	ایندهکس اولین جایی که کلمه مشخص شده در لیست پیدا می شود را در خروجی برمیگردداند
<u>insert()</u>	یک آیتم را در ایندهکس مشخص شده در لیست وارد می کند
<u>pop()</u>	یک آیتم از لیست را که ایندهکس آن مشخص شده در خروجی برگردانده و همچنین آن را حذف می کند
<u>remove()</u>	اولین جایی که کلمه ی مشخص شده در لیست پیدا می شود را حذف می کند.
<u>reverse()</u>	ترتیب آیتم های درون لیست را برعکس می کند.
<u>sort()</u>	یک لیست را به صورت صعودی (پیشفرض) مرتب می کند.

## Type of copy

- shallow copy
- deep copy

C6.

## C) Tuple

tuple protocols:

- tuple is a immutable data type.
- Tuple support indexing and slicing.
- because tuple is immutable data type, it uses optimal of memory

how to define tuple: C7.

packing and unpacking: C8.

List functions:

List methods:

<u>count()</u>	تعداد باری که یک آیتم مشخص در تاپل تکرار می شود را برمیگرداند.
<u>index()</u>	موقعیت اولین جایی که کلمه‌ی مورد نظر در تاپل پیدا می شود را در خروجی برمیگرداند

## None type

how to define None type: C14.

## Mutable – Immutable

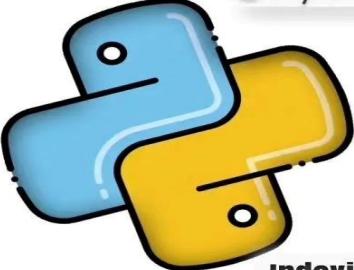
python may be have mutable or immutable objects.

- **mutable**

- you can edit it value without change its address.
    - List.
    - Dictionary.
    - Set.

- **Immutable**

- you can not edit its value without change its address.
    - Numbers.
    - Tuple.
    - String.
    - String frozenset [note: immutable version of the set]
    - Boolean.



# PYTHON DATA STRUCTURES

Indexing

Ordered

Mutable

Duplicate



List



Tuple



Set



Dictionary

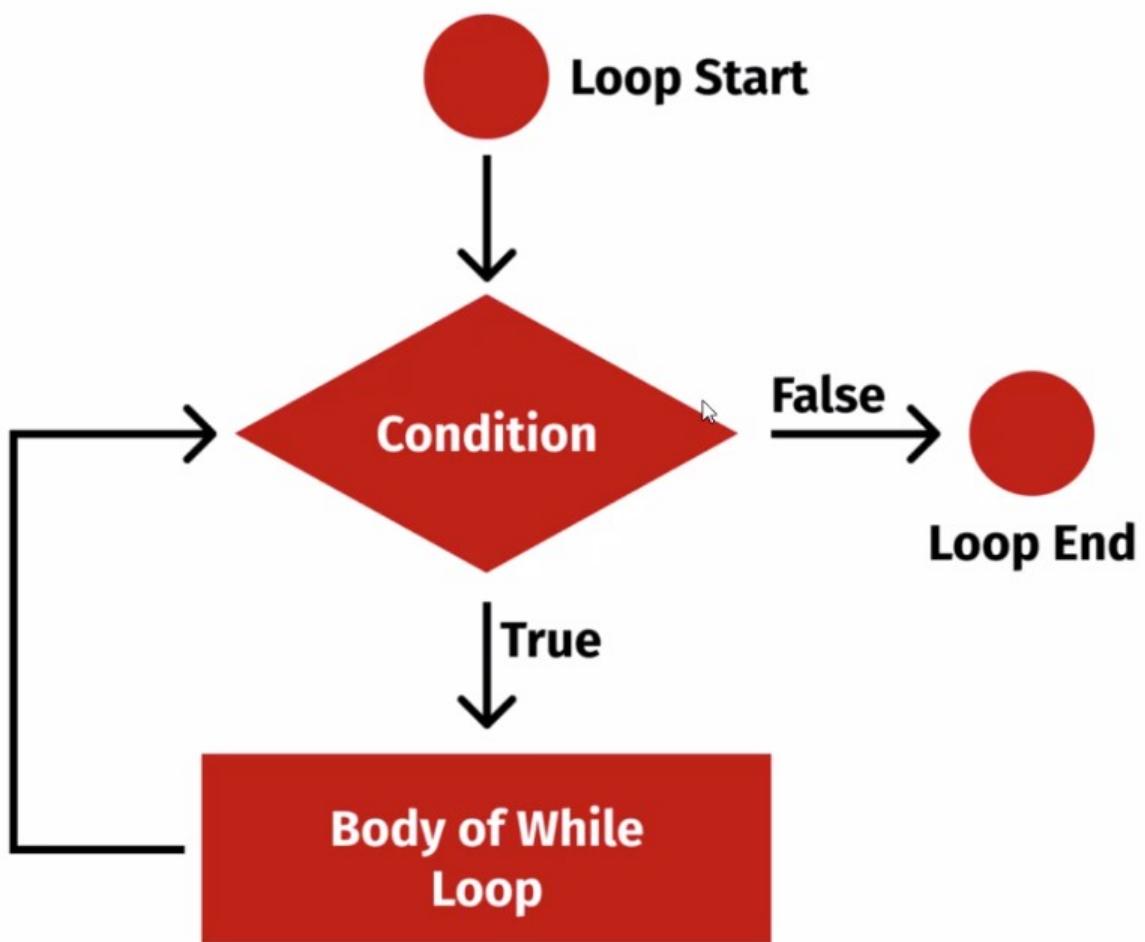


#### 4) Control commands(decision)

control commands:

- condition( تصمیم )
- repeat( تکرار )

## 5) Control commands(repeat)



[pythonpool.com](http://pythonpool.com)

## Break-continue-else in loop

: هر زمانی به این دستور در داخل حلقه رسید، به اصطلاح حلقه میشکن و میاد بیرون.

: هر زمانی به این دستور رسید ادامه دستورات بدن حلقه رو ول میکن و میاد ادامه چرخش حلقه رو ادامه میده.

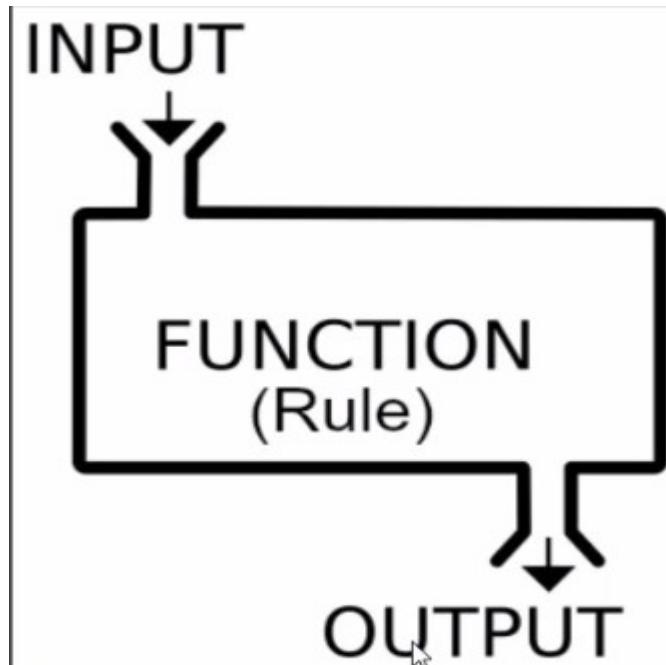
: زمانی بدنی این دستور اجرا میشه که حلقه بصورت نورمال به پایان برسه و دستوراتی مثل break یا خطای رخ نده.

## Loop technique

`enumerate(obj, start=3)`: this function returns every iterable elements in tuple with its index, **C16**.

## 6) Functions

function has input and output and convert input to output with processes.



# What is a function?

input ↓

rule

output ↓

$f(x) = 2x + 1$

x	y
-2	-3
-1	-1
0	1
1	3
2	5

domain      range

$f(x) = 2x + 1$

[mathycathy.com/blog](http://mathycathy.com/blog)

## Function name

getting function name:

```
function_name.__name__
```

## Parameter and argument

: ورودی هایی که موقع تعریف تابع تعریف شده‌اند پارامتر می‌گویند.

روش‌های پارامتر دهنده: .C20

پارامترهای نشانگر:

\*: هر زمان بین پارامترها از \* استفاده کردیم باید در آرگومان دهی آرگومان‌های بعدیش حتماً  
 بصورت name = value باشد .C22

\: هر زمان بین پارامترهای از / استفاده کردیم باید در آرگومان دهی آرگومان‌های قبلیش حتماً  
 بصورت موقعیتی باشد .C23

Args\*: in parameters, that is, receive as a Tuple.

Kwargs\*\*: in parameters, that is, receive as a Dictionary.

: ورودی هایی که موقع صدا زدن تابع مقداردهی می‌شوند آرگومان می‌گویند.  
روش‌های آرگومان دهنده: .C21

: یعنی ارسال محتوای args به عنوان آرگومان بصورت موقعیتی و به ترتیب پارامترها.  
\*\*: هر کلید دیکشنری رو به کلید هم نامش در پارامترها اختصاص میده.  
اگه موقع آرگومان دهی از استار استفاده نکنیم یعنی اولین پارامتر مقدار دهی بشه.  
.C21.1

## Documentation string

دک استرینگ، رشته‌ای که به عنوان اولین دستور در یک مازول، تابع، کلاس، و یا متده قرار می‌گیرد، است. کاربر باید در دک استرینگ بنویسد که یک تابع/کلاس و ... چه کاری انجام میدهد.

Define and access to doc string function: **C24**

doc string protocols:

- دک استرینگ شروع شود با حرف بزرگ با نکته تمام شود
- در خط اول خلاصه‌ای از کار تابع نوشته می‌شود
- در خط دوم اگر نیاز بود توضیحات بیشتر داده می‌شود
- و در خطهای بعدی ورودی و خروجی هاش مشخص می‌شود

مشخص کردن نوع parameter های تابع و خروجی های تابع در بخش **function annotation** سرایند **C25**.

از یک # برای نوشتتن توضیحات در رابطه با یک خط کد منطقی یا فیزیکی استفاده می‌شود.

دک استرینگ کلاس علاوه بر قوانین دک استرینگ توضیحات دیگری نیز درش می‌گنجاند. مثل مثالهایی از نمونه‌سازی از کلاس و استفاده از متدها میدهد **C105**.

ابزاری هست که با استفاده از اون میتوانیم مثالهایی که در کد نوشتمی رو تست کنیم. این ابزار کدهای نوشته شده در دک استرینگ رو تست می‌کنه تا ببینه که با نتیجه‌اش در دک یکسانه یا نه اگر یکسان بود که هیچ اگر نبود می‌گه که ما از این کد انتظار این نتیجه رو داشتیم ولی این نتیجه رو داده **C106**.

پایچارم خودش گودرتمند هست و خودش اگر یک پارامتری اگر مثلًا int بود str دادیم رو چک می‌کنه و می‌بینه که اگر یک پارامتری با نوع داده یکسان پاس نشده اینارو اطلاع میده. ولی در محیط هایی که این قابلیت پایچارم رو نداره برای این کار از mypy استفاده می‌شود.

mypy test1.py روش انجام: در ترمینال دستور زیر را اجرا می‌کنیم:

## First class

### function:

به تابعی first class میگیم که مثل همه آبجکت‌های دیگه به عنوان آرگومان بفرستیمیش یا از تابعی ریترن‌ش کنیم و یا به یک متغیری اختصاصش بدیم و ... بر خلاف زبان‌های دیگه توابع در پایتون بصورت دیفالت first class هستند.  
تابع second class هم داریم.  
**note**

### first class function attributes:

- Can be created at runtime

میتوانیم اون تابع first class رو داخل یک شرط، حلقه یا حتی داخل یک تابع دیگه بنویسیم تا در صورت اجرای همان بخش از کد تابع اجرا و ایجاد بشه و اگه اون تابع رو صدا زدیم و آن بخش از کد اجرا نشده بود در واقع انگار تابع تعریف نشده و خطای میده که تابع یافت نشد  
متلاً یک شرط برقرار نشد تابع اجرا و ایجاد نشه و انگار تعریف نشده C26.

- Can be assigned to a variable

f = func\_name

- Can be passed as an argument to a function C27
- Can be return as a result from a function C28
- Can have properties and methods

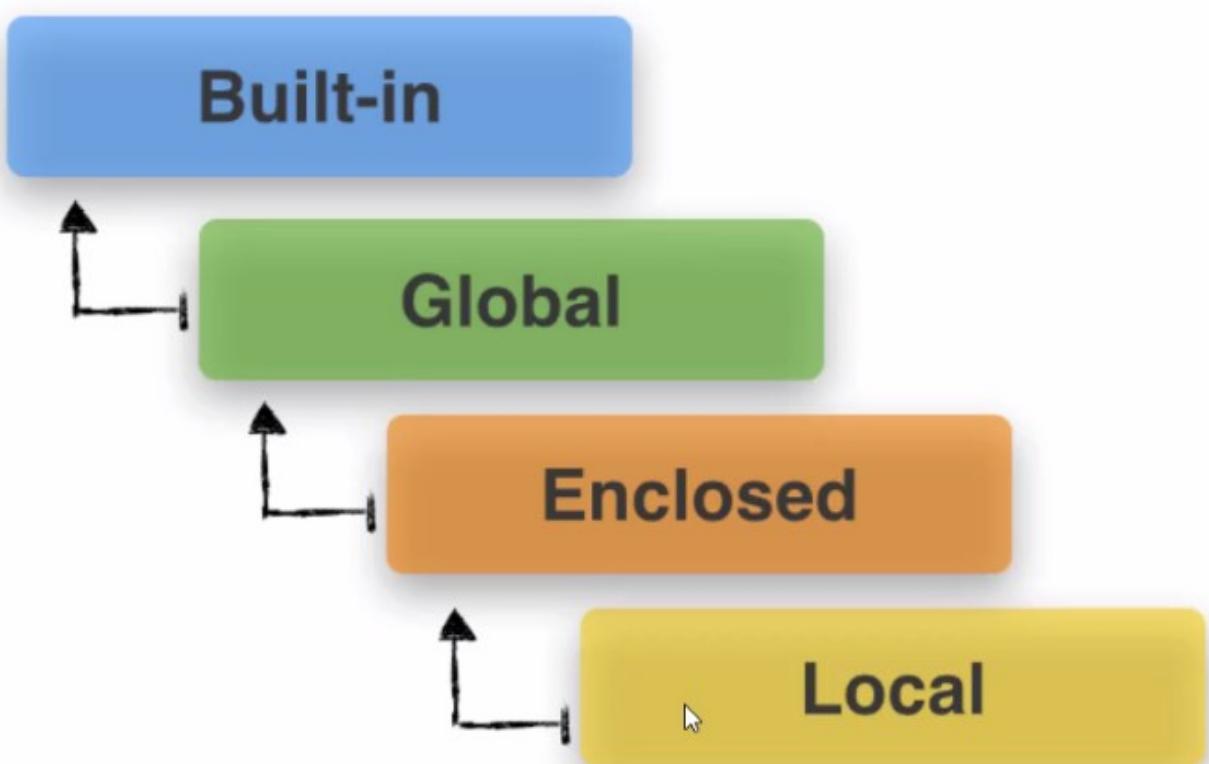
میتوانه attribute هایی هم داشته باشه.

## Name space – scope

**فضای نام:** اسامی افراد در هر خانواده منحصر به فرد هست و هیچ اسمی تکراری نیست ولی یک سطح از خانواده بالاتر شاید تو فامیل اسامی افراد شبیه هم باشه و انتخاب اسم منحصر به فرد برای هر شخص تو فامیل خیلی سخته و تو شهر نیز خیلی سخت تر و ... .

توی دنیای برنامه نویسی نیز سطوح مختلف داریم و توی یه پروژه کلی مازول و ابجکت هست که انتخاب اسم منحصر به فرد برای هر آبجکتی سخته ولی فضای نام که درواقع سطوحهای مختلفی رو در پروژه ایجاد میکنه و اسامی رو از هم جدا میکنه و اسامی در هر سطح منحصر به فرد میشن و باعث ایجاد اسامی تکراری نمیشن و اسامی هر سطح در دیکشنری های جداگانه ذخیره میشه.

سطوحهای مختلف فضای نام:



اعضای سطوح بالاتر همیشه در سطوح پایین‌تر قابل دسترسیه.  
فضای نام `Built-in`: فضای نام دیفالت و توکاری که از قبل وجود داره و اعضایی که در دیکشنری مربوطه‌اش ذخیره شده‌اند در همه فضاهای نام یا مازول‌های زیرینش در دسترس هست.  
تابع `print`, `len`, ... که در فضای نام `Built-in` هست در همه فضاهای نام پایینش قابل دسترسیه.  
.C29

فضای نامی که اسامی سطح یک `module or script` درش ذخیره شده Global.  
`Note`: در صورت نیاز به استفاده از اسامی تعریف شده در فضای نام Global یا مازول دیگر میتوانیم از اون فضای نام import کنیم.

زمانی فضای نام Enclosed ایجاد میشه که داخل یک تابع بنام A یک تابع دیگه بنام B تعریف بشه و در این صورت فضای نام تابع A فضای نام Enclosed خواهد بود و فضای نام تابع B فضای نام Local خواهد بود ولی اگر تابع تودرتو نباشد همون تابع A دارای فضای نام Local خواهد بود. C31

داخل یک `module or script` تابع‌های مختلفی میتونه تعریف بشه که اسامی داخل هر تابع در فضای نام همون تابع ذخیره میشه و درواقع برای هر تابع یک فضای نام Local ایجاد میکنه C32.

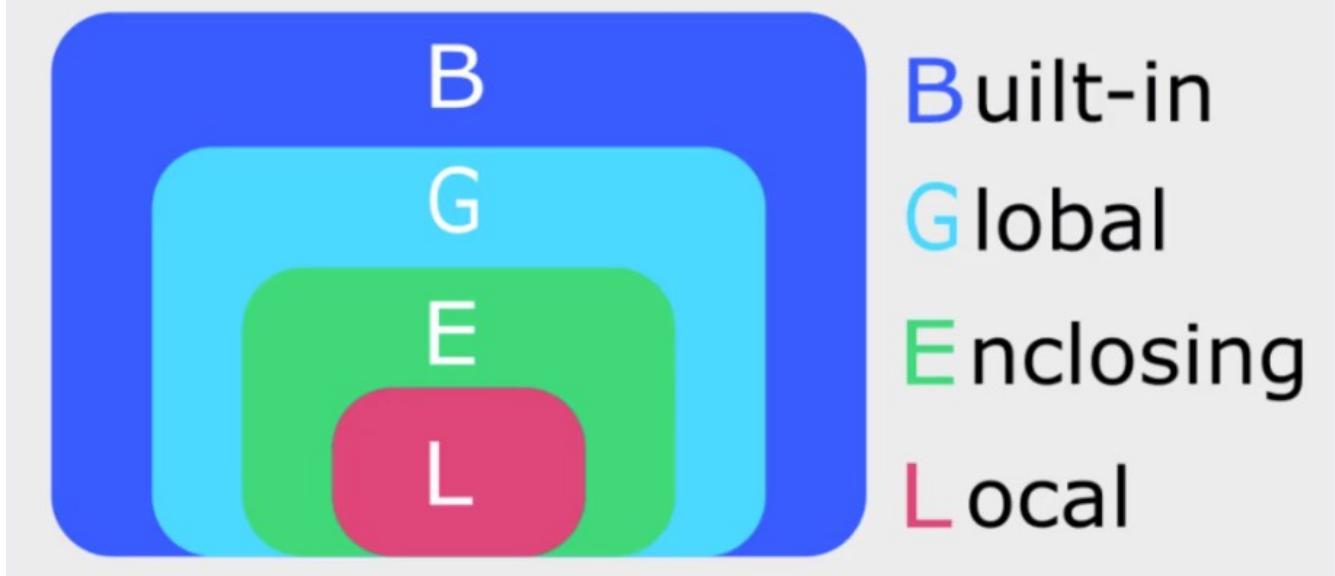
**حوزه:** توى يك بازى فوتبال كه دو تيم با همبازى ميکنن احتمالاً تو هر دو تيم اسم يك نفر رضا باشه و يك نفر از يك تيم اسم رضا رو صدا بزنە قانون حوزه اينو ميگه، ميگه اونى كه اسم رضا رو صدا زده اگر توى تيم خودش(حوزه مربوطه) اسم رضا بود احتمالاً با اون كار داره چون بهش نزديك تره و باید اون جواب بده ولی اگر توى تيم خودش رضا نبود احتمالاً رضاي تيم مقابل رو صدا زده و اگه تو تيم مقابل هم رضا نبود احتمالاً رضاي رو اسمش رو بردە كه تو باشگاهه و اگه تو باشگاه هم نبود احتمالاً رضاي كه تو شهر هست رو ميگه و ...

درواقع همان طور كه در تصویر زير هم مشاهده ميشه حوزه های مختلفی وجود داره كه مثلًا اگر اسم يك آبجکتی صدا زده شد ابتدا در نزديک ترین حوزه دنبالش ميگردد و اگه نبود ميره حوزه بعدی و به ترتيب از حوزه کوچيك به بزرگ دنبال اسم ميگردد و ... .

و دراقع وظيفه حوزه اين است كه موقع صدا زدن يك آبجکت اونو از فضای نام مربوطه پيدا كرده و دسترسی های لازم رو به استفاده کننده بدهد.

سطح های مختلف حوزه:

## Scopes in Python



Access to objects other scopes: C33.

**note:** دستورات کنترلی دارای فضای نام و حوزه نیستند.

در کل اسامی هر فضای نام در دیکشنری مختص خودش ذخیره شده و قوانین حوزه تعیین میکنه که اسامی ایی که در این دیکشنری‌ها ذخیره شده‌اند در کدوم حوزه‌ها قابل دسترسیه و یا در هنگام فراخوانی اولویت استفاده با کدومه.

## Pass by value – pass by reference

: در ارسال با مقدار فقط مقدار متغیر به تابع یا هر مقصود دیگر ارسال میشه و هر تغییری روی اون مقدار در مقصود مورد نظر اعمال بشه در مقداری که در مبدأ یا خارج از مقصود وجود داره تغییری اعمال نخواهد شد، و در حقیقت یک کپی از اون متغیر ارسال میشه .C34

: اما در ارسال با ارجاع برعکس ارسال با مقدار عمل میشه و درواقع در ارسال با ارجاع، تغییری که در مقصود روی مقدار اعمال میشه در مبدأ هم روی مقدار اعمال میشه. در حقیقت ارسال با ارجاع، رجوع کردن به متغیر هم همراه با مقدار ارسال میشه .C35

از اونجایی که همه چیز در پایتون شی هست پس نوع ارسال در هر شرایط بصورت دیفالت pass by reference هست ولی بستگی داره اون شی pass by value باشد یا نباشد، که اگر شیعی mutable or immutable باشد نوع ارسالش immutable هستن نوع ارسالشون mutable هست ولی اشیایی که

## Lambda

تابعی هست که در یک خط تعریف می‌شوند و بلعکس سایر توابع بدون نام هست و یک شی تابع return می‌کنه.

تعداد ورودی دلخواهی دارند و بدنده‌شون فقط دارای یک عبارت باید باشد و دستور return نیاز نیست زیرا این کار بصورت اتوماتیک انجام می‌شود، با کلمه کلیدی lambda شروع و قبل از : ورودی نوشته می‌شود و بعد از : عبارات، و عبارات نیز باید عبارات یک خطی باشند.

تابع‌های built-in پرکاربردی که لامبدا از آنها استفاده می‌کنند:

یکی یکی عناصر لیست را به تابع می‌فرسته و عملیات مورد نظر تابع روی عنصر اعمال می‌شود و سپس یک شی map می‌گرداند و اگر به لیست تبدیل شوند کنیم همون عنصرها با مقدار بروز شده را دریافت می‌کنیم.  
C36

یکی یکی عناصر لیست را به تابع می‌فرسته و عملیات مورد نظر تابع روی عنصر اعمال می‌شود و سپس یک شی filter می‌گرداند و اگر به لیست تبدیل شوند کنیم می‌بینیم که بر اساس اینکه خروجی تابع True or False هست لیست را فیلتر کرده، مثلاً در این مثال اعدادی که بزرگ‌تر از 20 هستند را فیلتر کرده و در لیست قرار داده.  
C37

مرتب سازی اشیا، که در این مثال حروف الفبا را بر اساس سایز آنها به کمک تابع length مرتب‌شون کرده  
C38

دوتا ورودی میگیره و عملیات مورد نظر رو روشون انجام میده و نتیجه اون دوتا رو با عنصر بعدی انجام میده و الی آخر، C39.

```
reduce(lambda x, y: x + y, [1, 2, 3, 4])
```

1	2	3	4
---	---	---	---

$$\begin{aligned} & \text{---} \\ & 1+2 = 3 \\ & \text{---} \\ & \text{---} \\ & 3 + 3 = 6 \\ & \text{---} \\ & \text{---} \\ & 6 + 4 = 10 \end{aligned}$$

## Iterator

: تکرار کردن.

: تکرار، از سرگیری.

دو تا ساختاری که در پایتون عمل iteration را انجام میدن لوب های while, for هستند.

: iterator در بکارگیری آنها در while و for

اگر با استفاده از while بخوایم عناصرهای یک iterable را به دست بیاریم باید قبلش عمل تبدیل به iterator را با متدهای \_\_iter\_\_ و سپس با متدهای \_\_next\_\_ در داخل حلقه while هر عنصر را به دست بیاریم ولی اگر این کار را با لوب for انجام بدیم این کارها بصورت اتوماتیک توسط خود for انجام میشه.

.(Lists, tuples, dictionaries, sets and so on) : iterable قابل تکرار، تکرار پذیر

به همهی شیهایی که بتونیم روی اونها عمل iteration را انجام بدیم iterable را انجام میدیم هستند. یا همه شیهایی که بتونیم اون هارو به iterator تبدیل کنم گفته میشه و همچنین هر شی iterable دارای متدهای iter است.

: Iterable methods

: اگر متدهای iter در خروجی های این تابع بود یعنی شی ورودی این تابع iterable هست.

: تکرارگر، تکرار کننده.

شیهایی هستند که آخرین وضعیت خودشون رو حفظ میکنند و با اجرای متدهای next عنصر بعدی رو میتوانند برگردانند. iterator ها علاوه بر اینکه متدهای \_\_iter\_\_ و \_\_next\_\_ را باید داشته باشند باید متدهای \_\_iter\_\_ و \_\_next\_\_ را هم باید داشته باشند که اصطلاحاً به این دو متدهای iterator protocol میگویند.

بعد از اینکه یک شی رو به iterator تبدیل کردیم میتوانیم متدهای next رو روش اجرا کنیم و با هر بار اجرایش عنصر بعدیش رو به دست بیاریم.

C40 از همین iterable میتوانه ایجاد بشه.

: Iterator methods

iter(x) or \_\_iter\_\_(x): convert iterable to iterator.

.`iterator` دریافت عنصر بعدی `next(x)` or `__next__(x)`

ها در هر بار `iteration` از اول شروع میشه ولی `iterator` ها از جایی که مونده. حالا میخوایم برای شی هایی که ایجاد میکنیم هم قابلیت `iterable`, `iterator` بودن رو پیاده کنیم. نحوه تبدیل `iterable` ساخته شده به `iterator`: وقتی متده `iter` رو کال میکنیم کارت `Iterable` رو تبدیل میکنه به `iterator` که متده `next` رو دارد.

.C40.1

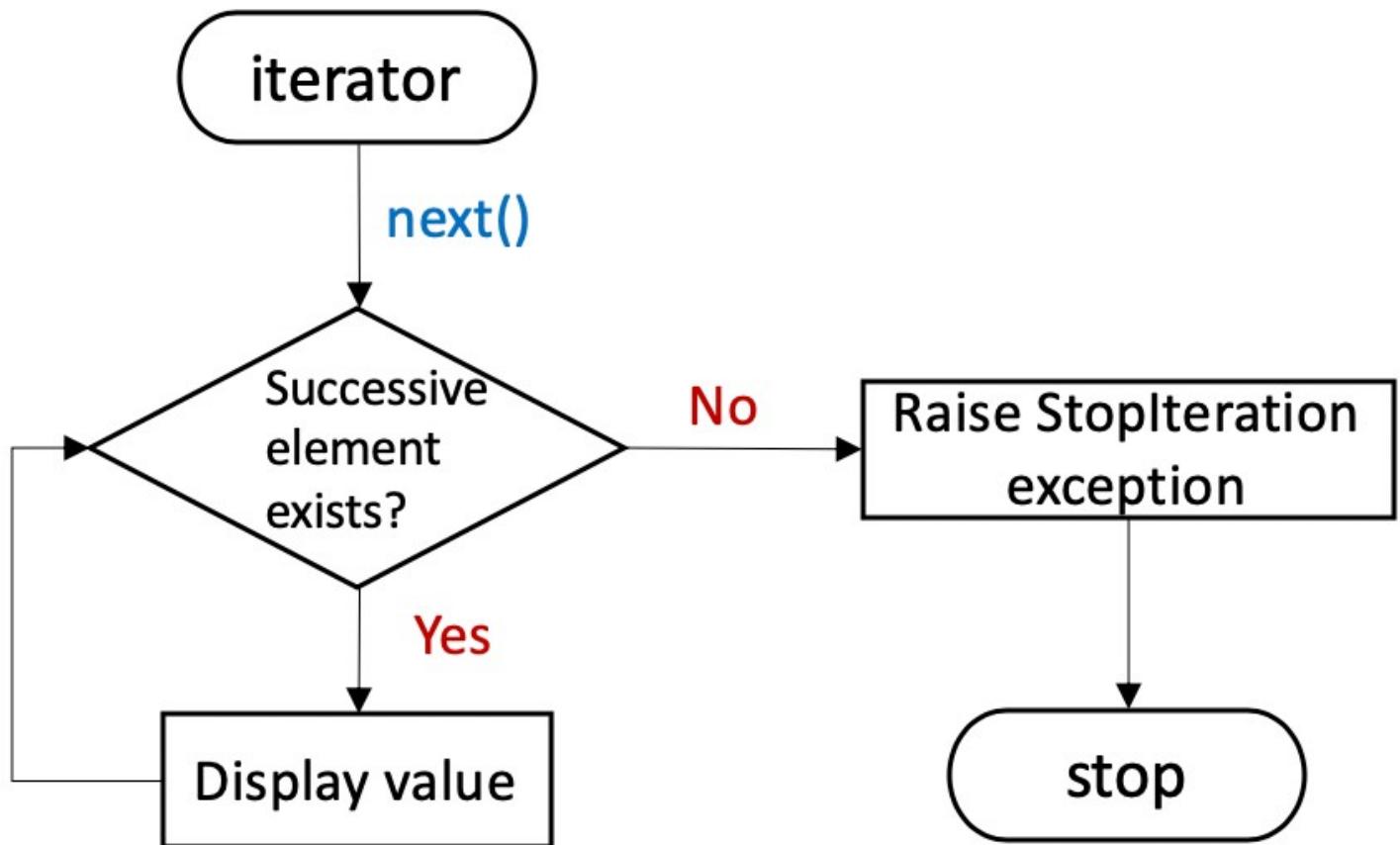
ولی میتوانیم با کال کردن خود متده `next` عنصر بعدی رو به دست بیاریم(از قابلیت `iterator` بودن استفاده کنیم)، بدون اینکه با متده `iter` به `iterator` تبدیلش کنیم .C40.2

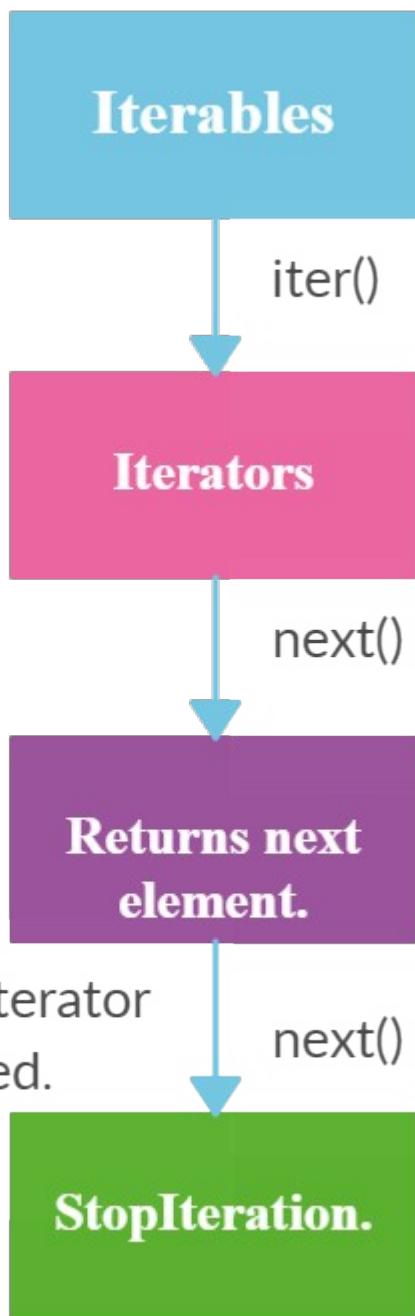
:ex

: در این مثال اگر متده `iter` را حذف کنیم و متده `next` رو کال کنیم، بدون مشکل کار میکنه و با هر بار کال عنصر بعدی رو ریترن میکنه.

ولی اگر متده `next` رو حذف کنیم و از متده `iter` استفاده کنیم خطأ رخ میده وقتی از متده `iter` استفاده میکنیم خود شی رو ریترن میکنه که خود شی هم، متده `next` رو داره و میتوانیم کالش کنیم تا به عناصر بعدی دست پیدا کنیم.

فیبوناچی- اعداد اول





## Generator

prerequisites: iterator.

(مولد) یا تولید کننده، هم خیلی تو پایتون پرکاربرد هست و کمک میکنه که کدهای ما خیلی بهینه باشند.

نحوهی عمل کرد ژنراتور: توی iterator ما یک شی iterable رو که عنصرهای مختلفی داره و از قبل آماده([L=1,2,3,...]) هست رو با استفاده از متده استفاده از متده iter به iterator تبدیل میکنیم و با استفاده از متده next به عنصرهای بعدیش دسترسی پیدا میکنیم ولی در ژنراتور برخلاف iterator که عنصرها از قبل آماده هست در اینجا از قبل آماده نیست بلکه عنصرهای بعدی توسط دستوراتی که ما داخل یک تابع پیاده میکنیم تولید میشه و به ما ارائه میشه.

در ژنراتور بجای کلمه کلیدی return از کلمه کلیدی yield استفاده میکنیم که تفاوتش با return این است که در تابع return رو صدا میزنیم تابع خاتمه پیدا میکنه و نتیجه تابع return میشه و داخل یک متغیر ذخیره میشه.

ولی وقتی از yield استفاده میکنیم تابع خاتمه پیدا نمیکنه بلکه اول یک مقداری با استفاده از کلمه کلیدی yield برمیگردونه(که یک عنصری هست که تولید شده) سپس stop میکنه و stop تازمانی ادامه پیدا میکنه که متده next دوباره اجرا بشه و دوباره این مراحل تکرار بشه و ژنراتور یک عنصر جدید تولید کنه و به عنوان یک شی generator یلد کنه و داخل یک متغیر save or print کنه .C49 اینکه هر عنصر با اجرای next در همان لحظه ایجاد میشه استفاده از منابع سیستم و ... خیلی بهینه میشه.

استفاده از ژنراتور بصورت تو در تو: در این کد ابتدا numbers به ژنراتور even\_gen داده میشه سپس ژنراتور even\_gen اعداد زوج رو yield میکنه و به عنوان ژنراتور به ورودی ژنراتور square\_gen داده میشه و سپس ژنراتور square\_gen توان دوهای yield های رو yield میکنه و به عنوان ژنراتور به ورودی لیست داده میشه و سپس print میشه .C50

## Generator methods:

: **close**: هر جایی این متدها زده بشه ژنراتور رو میبنده و انگار ژنراتور به انتهای رسیده و اگه بعد اجرای این متدها، متدهای **next** رو روی ژنراتور اجرا کنیم خطای **stopiteration** رخ خواهد داد.

: **throw**: با استفاده از این متدها میتوانیم یک استثنایی برای یک ژنراتور رخ بدمیم .**C51**  
: **send**

## Coroutine

(این توضیحات صرفاً برای توصیف رفتار کروتین در ژنراتور هست و گرنه بحث کروتین خیلی مفصله)

کروتین بصورت کلی یعنی دستورالعمل اتصال مجموعه ای از ورودی ها به مجموعه ای از خروجی ها یا به عبارت دیگر به عمل دریافت مقدار داخل تابع ژنراتور که توسط متدهای send خارج از تابع ارسال میشه کروتین گفته میشه.

برای راهنمایی ژنراتور و رساندن آن به اولین `yield` باید `next` باشد اگر کنیم سپس میتوانیم با متدهای `send` بهش مقدار ارسال کنیم و آن مقدار را بصورت `x=yield` دریافت کنیم و در داخل `x` قرار داده و استفاده کنیم. میتوانیم رفتار کروتین رو با `yield` بصورت ترکیبی (`get = yield x`) بنویسیم که در اینصورت عمل هر کدام انجام خواهد شد ولی اولویت اجرای کروتین دوم میشه. با هر بار اجرای متدهای `send` ژنراتور یک مرحله را طی میکند، انگار که متدهای `next` اجرا شده. و همچنین با پرینت کردن متدهای `send` (`print(g.send('test'))`) میتوانیم مقداری که `yield` برمیگرداند را به دست بیاریم [C52](#).

## Decorator

(تزيين گر) یا wrapper (پوشاننده یا پوشش دهنده) هم در کلاس‌ها و هم در توابع استفاده ميشه.

### decorator in function

بصورت ديفالت اگر تابع تودرتو داشته باشيم باید تابع داخلی توسيط تابع خارجي فراخوانده بشه .C41  
ولی اگر بخوايم اين توابع تودرتو رو با روش دكوراتور پياده کنيم باید تابع داخلی رو توسيط تابع خارجي  
بجای فراخوانی return کنيم.

در روش دكوراتور، تابع دكوراتور يك تابع ميگيره و تابع داخلی inner رو return ميکنه و در متغير d  
قرار ميده و هر موقع متغير d که اين بار تبديل شده به تابع inner رو که اگر اجرا کنيم علاوه بر اينکه  
يه کارهای ميتوونه قبل و بعد تابع func انجام بد، تابع func رو هم برای ما اجرا ميکنه و در حقيقت  
يعني تابع دكوراتور تابعی که از ما ميگره رو(func) برای ما بازنويسي يا تزيين ميکنه و اين ميشه  
دكوراتور.C42

New method use of decorator: C43.

Example for decorator:

(ex1) برنامه‌اي که با استفاده از دكوراتور جلوی خطای تقسيم بر صفر رو ميگيره: تابع division رو  
فراخوانی ميکnim و دو عدد که قراره عدد اولی بر دومی تقسيم بشه رو از ما ميگيره ولی چون امكان  
داره عدد دومی 0 وارد بشه پس با استفاده از دكوراتور جلوی خطای گرفته ميشه و قبل از اجرا تابع  
تابع دكوراتور(dec) ران ميشه و دو عدد داده شده به تابع inner داده ميشه و شرط 0 بودن  
يا نبودنش بررسی ميشه و سپس اگر نياز بود تابع func اجرا و دو عدد گرفته شده(x,y) بهش داده  
ميشه تا به عنوان ورودی به تابع division بده و اونو اجرا کنه و سپس تابع division اجرا بشه و  
نتيجه در همان جا که func صدا زده شده return کنه و در همان جا هم به بيرون از تابع که تابع  
division صدا زده شده return ميشه .C44

(ex2) اجرای متند next ژنراتور با دكوراتور .C53

در پياده‌سازی دكوراتور باید سعی بر اين باشه که ورودی های تابع inner بصورت (\*args, \*\*kwargs)  
باشه تا تابع دكوراتور بتوانه برای اکثر توابع دكوراتور باشه.

: در دکوراتور تودرتو، ترتیب اجرای دکوراتور اگر با روش قدیمی باشد از بیرونی ترین پارانتز شروع میکند و تا داخلی ترین پارانتز اجرا میکند و در روش فراخوانی جدید از بالاترین فراخوانی شروع میکند و تا پایین ترین فراخوانی اجرا میکند .C45

: اگر سه تابع تودرتو داشته باشیم میتوانیم اگر نیاز شد ورودی از طریق فراخوانی دکوراتور به بیرونی ترین تابع(dec\_equal) بدیم و ازش استفاده کنیم .C46

جلوگیری از بین رفتن اطلاعات تابعی که دکوراتور روش اعمال شده: بصورت دیفالت تابعی که دکوراتور روش اعمال میشه برخی اصلاحاتش ازجمله نام و doc string اش از بین میره و به جاش اطلاعات تابع داخلی inner که مال دکوراتور هست ریپلیس میشه ، ولی با استفاده از یک پکیجی بنام functools میتوانیم از این عمل جلوگیری کنیم.

نحوه استفاده از این پکیج به این صورت هست که از بعد اینکه import اش کردیم میایم بعد سرآیند تابع دکوراتور بصورت یک دکوراتور فراخوانیش میکنیم و func رو بهش میدیم.

در واقع پیکیج functools یک کپی از اصلاحات تابعی که قراره دکوراتور روش اعمال بشه(p\_name) از طریق func ایی که بهش میدیم میگیره و وقتی که نیاز داشتیم در اختیار ما قرار میده .C47

فرمول کلی پیاده‌سازی خود دکوراتور: C48

## Decorator in classes

دکوراتور کلاس هیچ فرقی با دکوراتور تابع نداره و فقط بجای اینکه `func` رو بگیره و رانش کنه و نتیجه تابع رو ریترن کنه `cls` رو میگیره و رانش میکنه (instantiation) و نمونه ساخته شده رو ریترن میکنه. حتی میتوانیم دکوراتوری که ساختیم رو رو متدهای کلاس هم اعمال کنیم.

نحوه پیاده‌سازی دکوراتور با کلاس: برای اینکه با استفاده از کلاس دکوراتور بسازیم باید دو تا متدهای `init` و `call` حتماً پیاده‌سازی بشن. توی اینیت `func/cls` به عنوان ورودی گرفته و مقدار دهنده میشه و توی `call` اون کارهایی که قراره قبل و بعد از اجرای `func/cls` انجام بشه رو پیاده میکنیم. بجای `wraps` از `update_wrapper` استفاده میکنیم.

.C48.1

## Function attributes

تو دنیای برنامه نویسی شی گرا، برنامه ما از یک سری واحد ها تشکیل میشے به اسم کلاس، هر کلاس متدها و صفات(نام دانشجو، ش دانشجویی، و...) مختلفی داره.

تابع هم مثل بقیه اشیاء در پایتون یک شی هست و صفات(`docstring` تابع(-`.name`-)، نام تابع(-`.doc`-)) مختلفی داره.

مثلاً اگر یک تابع داشته باشیم برای محاسبه تعداد چرخش میلنگ یک ماشین در کارخانه ایران کاوه که بصورت دیفالت صفات نام و داک استرینگ رو داره و علاوه برآونها میتونیم صفات نام کارخانه، نوع ماشین، نوع موتور و... برای اون تابع بعنوان صفات اضافه کنیم [C54](#).

## Attribute functions:

- : اضافه کردن صفات به تابع با توابع داخلی پایتون.  
`setattr(obj, "attrName", "attrValue")`
- : دریافت صفات تابع با توابع داخلی پایتون.  
`getattr(obj,"attrName","attrDefVal")`
- : چک کردن اینکه آیا یک تابعی دارای یک صفتی است یا نه.  
`hasattr(obj, "attrName")`
- : حذف یک صفت.  
`delattr(obj, "attrName")`

## Attribute:

- : دریافت documentation string تابع مورد نظر.  
`--doc--`
- : دریافت نام تابع.  
`--name--`
- : دریافت صفات تابع در یک دیکشنری.  
`--dict--`

## Recursive function

تابع بازگشتی یک روش برای حل مسله‌ای است که نیاز به تکرار داره و این تکرار با فراخوانی مجدد تابع انجام میشه.

تابع بازگشتی از ساختار stack استفاده میکنه.

```
def recursive():
    ...
    if stop_condition:
        ...
        return ...
    ...
    recursive()
```

```
recursive()
```

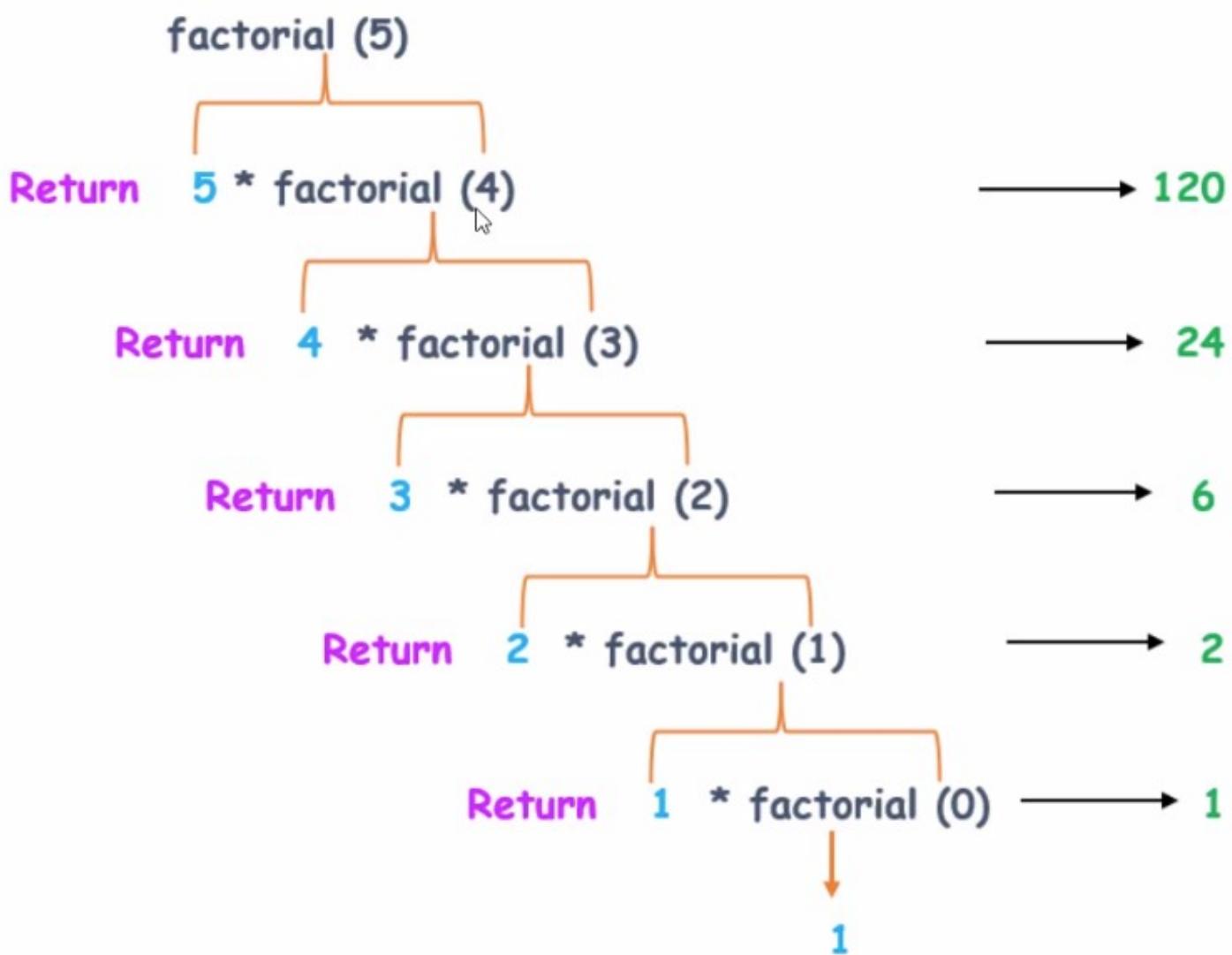
نکاتی که در پیاده‌سازی تابع بازگشتی باید رعایت شود:

مشخص کردن شرط توقف تابع بازگشتی.

مشخص کردن فراخوانی دوباره تابع بازگشتی.

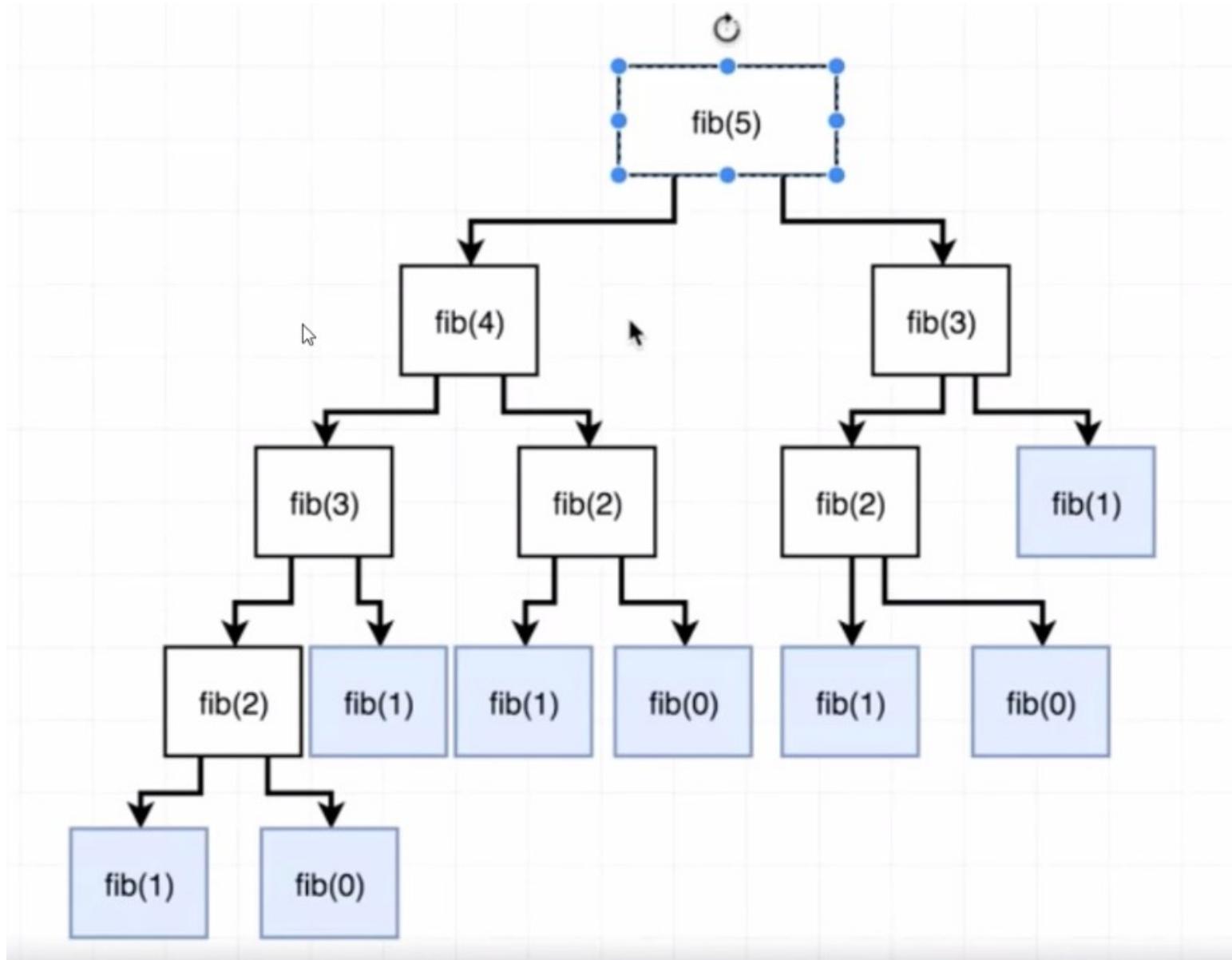
Example for recursive function:

(ex1) برنامه محاسبه فاکتوریل: فرض کنید از تابع میخوایم فاکتوریل 3 را برای ما حساب کنه؛ اول برای اینکه فاکتوریل 3 رو پیدا کنه 3 رو باید ضرب کنه به فاکتوریل 2 و چون فاکتوریل 2 مشخص نیست میره فاکتوریل 1 رو پیدا کنه و بازم چون فاکتوریل 1 مشخص نیست میره فاکتوریل 0 رو پیدا کنه که برای فاکتوریل 0 با شرط return مشخص کردیم عدد 1 رو برگردونه پس عدد 1 برگردانده میشه و سپس فاکتوریل 1 حساب و میشه و سپس فاکتوریل 2 حساب و return میشه و سپس فاکتوریل 3 حساب و return میشه و تمام .C55



(ex2) تابع بازگشتی که حاصل ضرب اعداد بزرگ‌تر از 5 رو محاسبه میکنه C56

ex3) تابع بازگشتی سری فیبوناچی C57.



## Decorator and recursive generator function - recursive depth

اعمال کردن دکوراتور روی تابع بازگشتی دکوراتور

اعمال کنیم در هر مرحله که تابع بازگشتی مجدداً خودش رو فراخوانی میکنه دکوراتور هم مجدداً روش اعمال میشه مثلاً در این مثال که دکوراتور بر روی تابع `_timer` اعمال شده در هر مرحله که تابع `_timer` داخلش خودشو فراخوانی میکنه در هر فراخوانی مجدداً دکوراتور روش اعمال میشه [C58](#).

ژنراتور بازگشتی: همان‌طور که قبلاً اشاره شده تابع ژنراتور، شی ژنراتور برمیگردونه، پس اگر تابع ژنراتور رو بصورت بازگشتی بنویسیم تابع در اولین `yield` متغیر `n` برمیگردونه ولی چون میخوایم بصورت بازگشتی باشه و هر عنصر رو برگردونه پس تابع ژنراتور رو فراخوانی میکنیم ولی چون فراخوانی تابع شی ژنراتور برمیگردونه پس با حلقه هر عنصرش رو `yield` میکنیم [C59](#).

عمق بازگشت: تعداد فراخوانی هایی که یک تابع بازگشتی میتونه فراخوانی بشه و هر بار که تابع بازگشتی خودش خودشو فراخوانی میکنه این فراخوانی ها تو پشته ذخیره میشه.

بصورت دیفالت عمق بازگشت هر تابع بازگشتی تا 1000 بازگشت هست ولی میتوانیم عمق بازگشت رو تغیر بدیم:

`getrecursionlimit`: دریافت عمق بازگشت.

`setrecursionlimit`: سُت کردن مقدار عمق بازگشت.

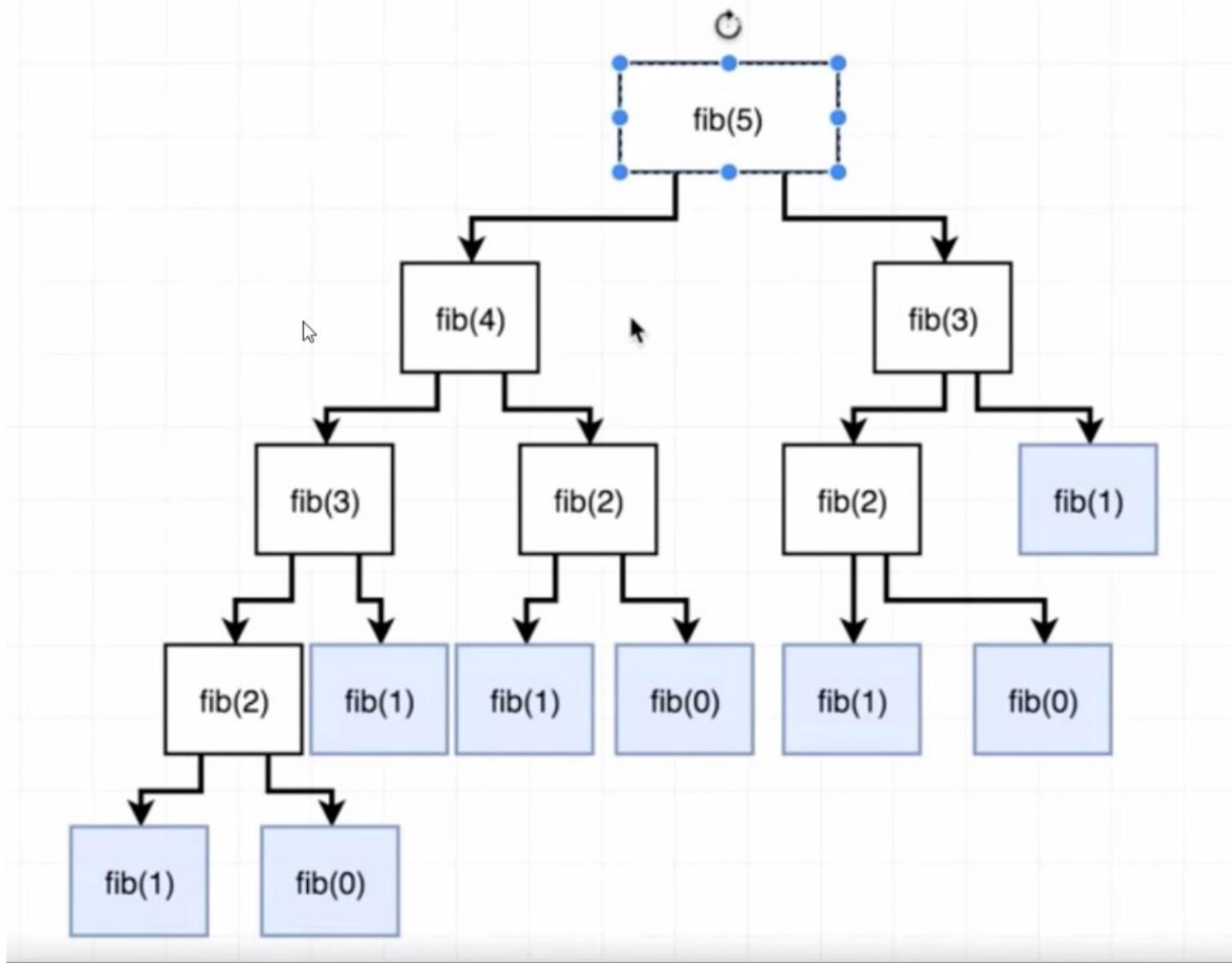
زمانی به درد میخوره که حافظه‌ی محدودی داریم یا شک داریم که احتمالاً تابع بازگشتی وارد فراخوانی بی‌نهایت بشه و ... .

**Note**: مثلاً عمق بازگشت رو 15 سُت کردیم و انتظار داریم 15 بار تابع بازگشتی فراخوانی بشه ولی `n` بار کمتر فراخوانی میشه، این برای اینه که این فراخوانی ها در پشته ذخیره میشه و ممکنه لا به لای این فراخوانی ها `n` تا مربوط به عملیات خود تابع بازگشتی ذخیره بشه و `n` تا از 15 کمتر فراخوانی بشه.

و اگر تعداد فراخوانی ها  $+ n$  بیشتر از 15 بشه خطأ رخ میده.

[C60](#)

## Memoization technique



تابع بازگشته ممکنه کارهای تکراری انجام بده که باعث کندی تابع میشه و میتوانیم با استفاده از تکنیک "memoization" جلوی کارهای تکراری رو بگیریم.

مثلاً در تابع بازگشته فیبوناچی، اگر به تصویر بالا توجه کنیم متوجه میشیم که بعضی از محاسبه ها چند بار تکرار شده، مثلاً محاسبه های 3,2 چند بار محاسبه شده و این تعداد محاسبه یا انجام کار تکراری در فیبوناچی 5 که عدد کمی هست قابل چشم پوشیه ولی اگر فیبوناچی عدد خیلی بزرگ رو از تابع بازگشته بخوایم این محاسبه یا کارهای تکراری خیلی زیاد میشه که باعث میشه تابع هنگام اجرا زمان زیادی مصرف کنه و ... .

: تکنیکی هست که با استفاده از تابع دکوراتور اختصاصی هر تابع نوشته میشه.

نحوه عمل کرد تکنیک به خاطرسپاری به این صورت هست که یک دکوراتور روی تابع مورد نظر اعمال میکنیم که این دکوراتور در هر بار قبل از فراخوانی تابع بازگشتی fib، اول چک میکنه که آیا این ورودی که میخوای بدی به func تا بره فیبوناچی شو پیدا کنه، فیبوناچیش داخل دیکشنری memory نیست؟ اگر نیست؟ میره تابع بازگشتی رو اجرا میکنه سپس نتیجه تابع بازگشتی رو با کلید n که ورودی تابع هست به دیکشنری memory اد میکنه و سپس اون کلید رو return میکنه ولی اگر n داخل اون دیکشنری باشه دیگه تابع بازگشتی رو اجرا نمیکنه و میره از دیکشنری عنصر n رو return میکنه و این باعث صرفه جویی در زمان میشه.

در حقیقت فیبوناچی هایی که قبلاً محاسبه شده‌اند و در memory ذخیره شده‌اند رو دوباره محاسبه نمیکنه و از memory برمیداره C61.

## 7) Comprehension

ادراک یا خلاصه سازی که کارش خلاصه سازی یا کوتاه سازی کد در ایجاد نوع داده‌های مختلف از جمله list و مخصوصاً dictionary, set, generator

### List Comprehension

Syntax: در کل comprehension سعی میکنه که کد هارو ساده و کمتر کنه.  
Note: در کد معمولی اول شرط‌ها و حلقه‌ها نوشته میشه و سپس اون عنصری که قراره به لیست اد بشه آخر همه دستورات نوشته میشه.

ولی در روش comprehension افزودن به لیست اول همه دستورات نوشته میشه و شرط‌ها و حلقه‌ها سپس نوشته میشه و چون بلعکس نوشته شده دلیل بر این نیست که اولویت اجرا نیز بلعکس شده بلکه اولویت اجرا مثل کد معمولی هست

یعنی مثل کد معمولی اگر همه شرط‌ها و حلقه‌ها درست اجرا شدند سپس عنصر به لیست اد میشه.

در تصویر زیر کد معمولی با ترتیب در comprehension جایگذاری شده ولی ممکنه بعضی ترتیب عوض بشه مثل مثال 3.

The diagram illustrates the conversion of a nested loop comprehension into its corresponding normal loop code. On the right, a screenshot of a Python code editor shows a nested comprehension:

```
l = []
for sub1 in iterable:
    if cond1(sub1):
        if cond2(sub1):
            for sub2 in sub1:
                if cond3(sub2):
                    for elem in sub2:
                        l.append(func(elem))
```

On the left, the equivalent normal loop code is shown:

```
[func(elem) for sub1 in iterable if cond1(sub1) if cond2(sub1) for sub2 in sub1 if cond3(sub2) for elem in sub2]
```

Arrows from the nested comprehension code point to specific parts of the normal loop code, indicating how each part maps to the corresponding structure in the comprehension. The normal loop code uses multiple levels of indentation to represent the nested loops and conditionals.

```
x = [i * 2 for i in range(10)]
```

: هر چیزی که قراره به عنوان یک عنصر بهش اد بشه.  
for i in range(10) : هر دستوری که مقدار تولید میکنه.

Example for comprehension:

- .C66 : مثالی که یک عنصر  $|1$  را با هر عنصر  $|2$  در یک تاپل به لیست اد میکنه (ex1)
- .C67 : مثالی که کاراکترهای هر اسم رو به لیست اد میکنه (ex2)
- .C68 : برعکس کردن سطر وستون ماتریکس (ex3)

:Note

.C69 برای خودش حوزه ای مستقل دارد • comprehension

## Generator Expression

:Note

- منطقاً چون در لیست میگیم generator در ژنراتور هم باید بگیم list comprehension استثناء بهش میگن comprehension .generator expression برای ژنراتور اکسپرشن از () استفاده میشه.

### Generator expression examples:

.C70) سینتکس ژنراتور اکسپرشن: ژنراتور اکسپرشنی که یه رنجی رو به ما میده

## Set Comprehension

Set comprehension examples:

.C71 ای که یه رنجی رو به ما میده :set comprehension و set comprehension سینتکس (ex1)

## Dictionary comprehension

### Dictionary comprehension examples:

سینتکس dictionary comprehension ایی که دو تا لیست (ex1) و :dictionary comprehension میده سپس ازش key, value را برمیداره و یک دیکشنری جدید درست میکنه .C72

## 8) Modular programming

ایده‌ی اصلی برنامه نویسی ماژولار، تقسیم وظایف یک برنامه بزرگ به چندین زیربرنامه‌ی کوچکتر و مستقل است.

ماژولارسازی نرم افزار برای درک بهتر برنامه نویسان و محدودیت‌های شناختی انسان‌ها انجام می‌شود که مجبور هستند برای خوانایی، کیفیت، نگهداری و اشکال‌زدایی بهتر برنامه‌ی خود، کدها را به قطعات کوچک‌تری تبدیل کنند در حالی که کامپیوترها برای درک کد نیازی به تقسیم آن به بخش‌های کوچکتر ندارند.

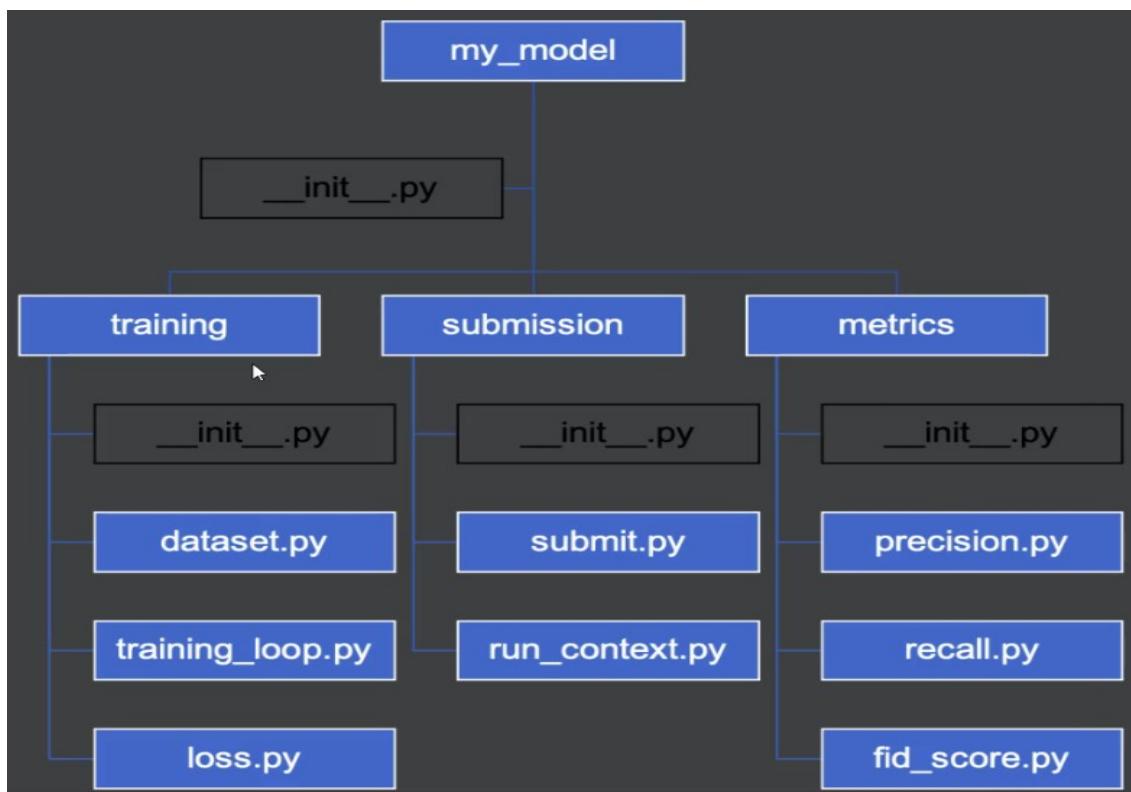
نحوه تشخیص کدهایی که بهتره بصورت ماژولار نوشته بشن: آیا این کدی که الان دارم مینویسم ممکنه برای بخش دیگه‌ای از برنامه هم استفاده بشه؟ اگه جواب مثبت بود، یعنی این کد باید ماژول بشه.

مزایای برنامه نویسی ماژولار:

- اشکال‌زدایی آسانتر
- قابلیت استفاده‌ی مجدد
- خوانایی کد
- قابلیت اطمینان
- قابلیت نگهداری
- کیفیت و تست نرم افزار
- برنامه نویسی گروهی

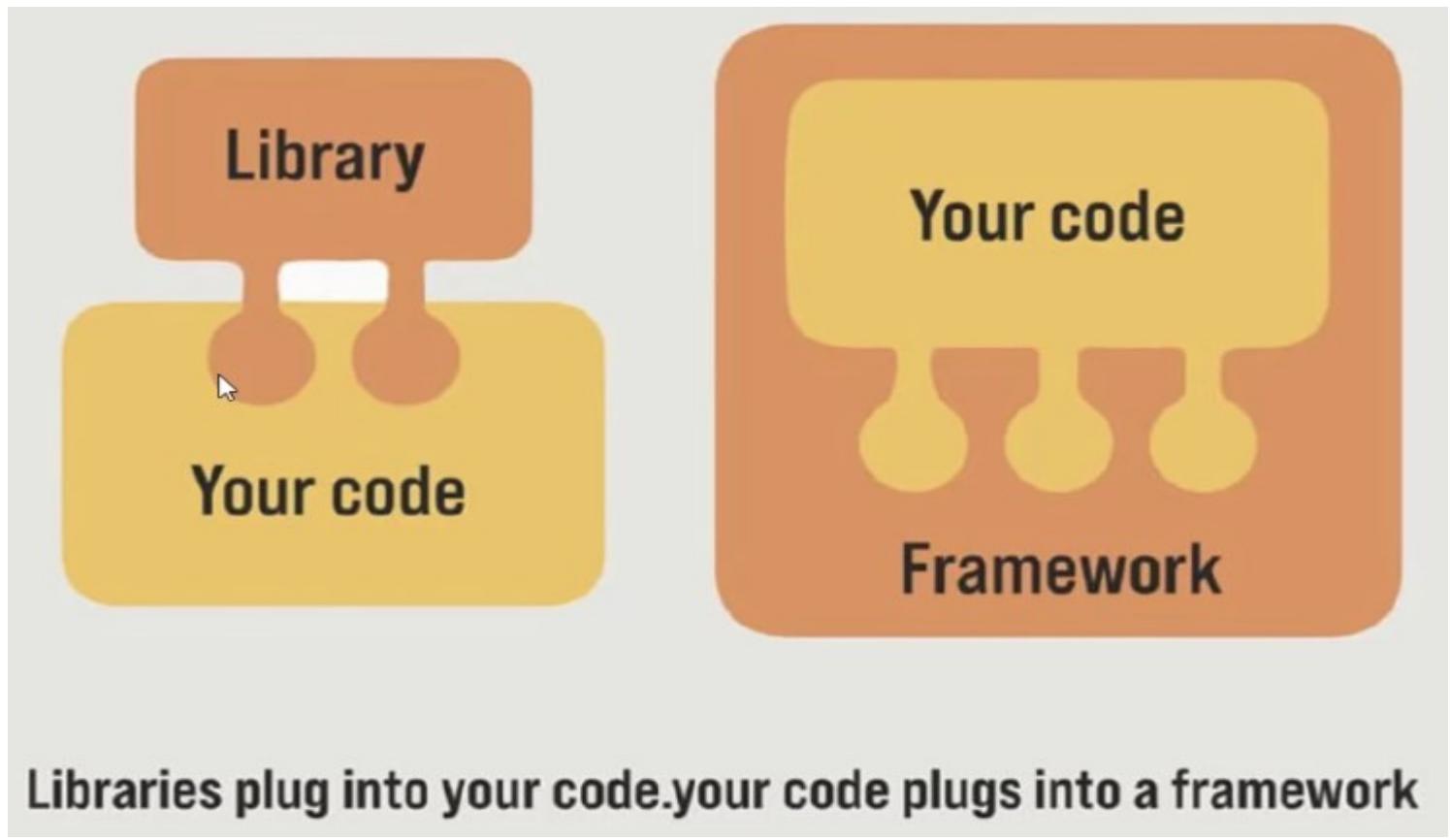
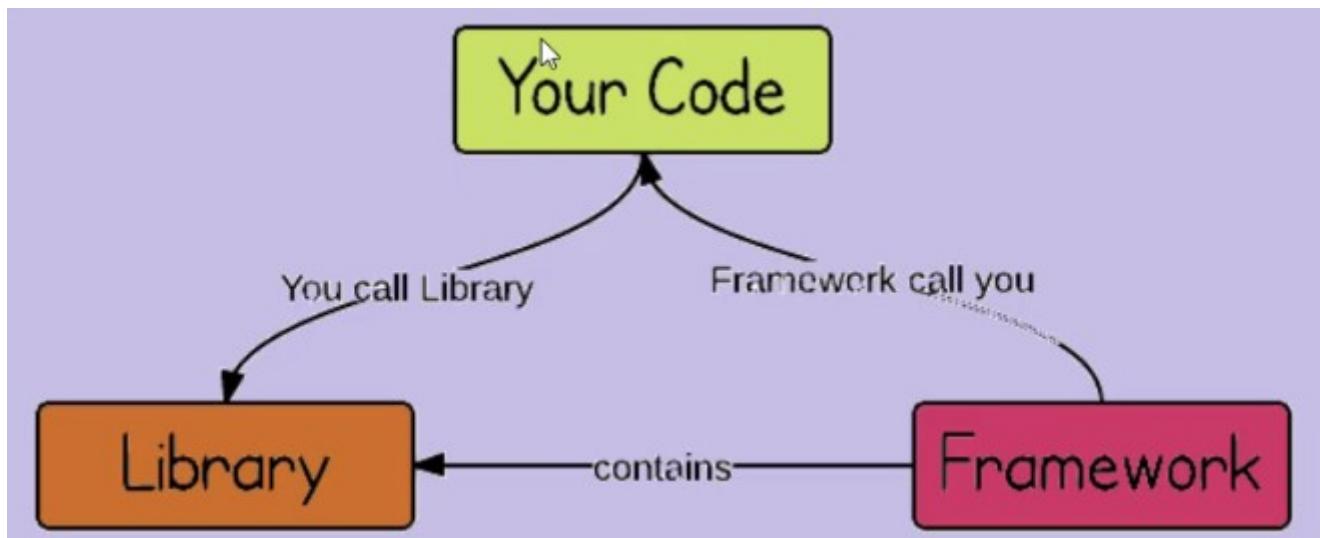
## Concepts of script, module, package, library, framework

- **Script**: فایل .py ایی که توش یه سری دستورات نوشته شده برای انجام یک کار خاصی و بصورت مستقیم run میشه.
- **Module**: ماژول به تنها یی کاربردی ندارد بلکه مکملی هست برای برنامه های دیگه.
- **Pure modules**: فایل .py ایی که خودش به تنها یی run نمیشه بلکه توسط script های دیگه ای یا فایل های py دیگه ای import و استفاده میشه و معمولاً از کلاس ها و توابع درش استفاده شده.
- **Extension modules**: ماژول هایی که به زبان های دیگه ای نوشته شده و در داخل پایتون استفاده میشه.
- **Note**: از script میتوان بعنوان module استفاده کرد و بلعکس، در این صورت هر script را میتوان module و هر module را script نامید.
- اگر فایل .py ایی رو به تنها یی اجرا کنیم میشه script ولی توی یه فایل دیگه import و استفاده کنیم میشه .module میشه.
- **Package**: مجموعه ای از modules مرتبط و شاید وابسته به هم به همراه فایلی بنام initialization (\_init\_\_.py) در داخل یک directory است. پکیج ها میتوون بصورت تودر تو نیز باشنند.



: `initialization(__init__.py)` یسری دستورات پیشنهادی مثل مقدار دهی های اولیه، صدا زدن یسربی متدها و ... میتوانه نوشته بشه که تمام دستوراتی که داخل `init` نوشته شده با `import` کردن ماژول مورد نظر اجرا خواهد شد.

- `Library` به ترکیب چند تا پکیج باهم رو در زبان های دیگه لایبرری می نامند.
- در پایتون کم کاربرد دارد و تقریباً میشه گفت معادل آن در پایتون همان `package` هست.
- معمولًا pakage بزرگتر از library هست.



## Building and use the Module and Package

معنی برخی از فایل ها:

: برای پیکربندی پروژه استفاده میشه.

: دستورات مربوط به خط فرمان و همچنین یک رابط برای خط فرمان هستش.

: توصیف و توضیحاتی در مورد پروژه.

: معرفی فایل های غیر پایتونی که تو پروژه استفاده شده.

: پیشنبازهای پروژه رو معرفی میکنه مثلًا میگه فلان پکیج ها یا فایل ها برای اجرای این پروژه نیازه.

: پروانه انتشاره پروژه هستش.

: داکیومنت پروژه هستش و راهنمایی، آموزش و توضیحاتی رو در رابطه با پروژه ارائه میده.

: تست های پروژه درش قرار میگیره.

: فایل های .pyc کامپایل شده ی مازول هایی که import شده اند داخلش قرار میگیره که در هر بار استفاده دسترسی سریع به مازول مورد نظر possible باشه.

انواع حالت های استفاده از مازول:

- **حالت اول:** یه سری مازول ها بصورت دیفالت روی خود پایتون وجود داره و برای استفاده از آنها، باید مازول مورد نظر رو import کرد.
- **حالت دوم:** یه سری مازول هایی رو یه سری از برنامه نویس هایی نوشتن و مثل مازول های دیفالت پایتون برای همه پرکاربرد نیست و برای استفاده از آن، باید مازول مورد نظر رو توسط package manager در virtual enviroment نصب و import کرد.  
در این حالت قبل از اجرای دستور نصب، باید فایل activate رو run کرد تا پیکچهای نصبی در virtual enviroment نصب شوند.
- **حالت سوم:** ایجاد مازول سفارشی.

## Module import methods:

- `import(import Package or Module)`
  - `import moduleName as mn`
- `from(import Module or Package or nameX(func, class, var , etc))`
  - `form moduleName import x,y:`
  - `form moduleName1.moduleName2 import moduleOrPackageName3: moduleNameOrPackageName3.nameX`
  - `form moduleName import *`: Imports all names in the module except private names(\_x(func, var, etc) : names that are private)).

all: اگر متغیری با این نام داشته باشیم موقع `import` کردن بصورت \* فقط اسمی یا مازول هایی که عضو این لیست هستند `import` خواهند شد.

**Note**: اگر از \* در `import` کردن استفاده کنیم و اگر `var` ای بنا نباشه همه‌ی اسمی استفاده کردن `nameX(func, class, var , etc)` خواهد شد بجز `private, protected` ها و اگر متغیر `_all` را تعریف کنیم در روش \*، فقط اونها یی که در متغیر هستند `import` میشن حتی اگر `private or protected` هم باشند. و برای `import` کردن `module or packages` بصورت \* تعریف `_all` اجباری هست. معایب این روش:

(1) اسمی بلا استفاده وارد میش.

(2) ریشه اسمی در بین چند صد کد مشخص نمیشه. یعنی نمیتوانی بفهمی که از کدام مازول/پکیج وارد شده.

(3) تداخل نام با نام های سایر پکیج ها.

- `from .test import f:`

اگر از . در داخل فایل a برای مسیر دهی استفاده کنیم و همون فایل a رو مستقیم `run` کنیم خطای `ImportError` میده. چون در هنگام اجرای یک فایل بصورت مستقیم `filename = _main` و مفسر `filename` توانایی تشخیص `parent` فایل رو نداره.

ولی اگر فایل a رو از داخل فایل b ران کنیم و چون در این صورت `filename = a` هست، پس در این صورت `parent` فایل تشخیص داده میشه و خطای `ImportError` رخ نخواهد داد.

پس در صورتی از . در مسیر دهی استفاده میکنیم که فایل رو بصورت مازول `import` کنیم. کوتاه کردن کد/مسیر `:import`

```
from parent.child1.m1 import MONTH, Animal
```

این مسیر طولانیه:

برای کوتاه کردنش در فایل `__init__.py` اسامی مورد نظر را بصورت زیر وارد میکنیم

```
from .child1.m1 import MONTH, Animal
```

سپس در جایی که میخوایم بصورت کوتاه `import` میکنیم:

```
from parent import MONTH, Animal
```

چرا بجای بعضی از کلاس‌ها باید شئی از اون کلاس را `import` کنیم: بعضی از کلاس‌ها که فقط به یک نمونه از اون کلاس نیاز داریم به همین دلیل همون یدونه شی رو در ماثولی که کلاس هست میسازیم و بجای خود کلاس شئی که از اون کلاس ساخته شده رو `import` میکنیم. مثل کلاس اتصال به db.

ولی این روش یک مشکل داره اونم اینه که:

در حین `import` کردن، شئ ساخته و سپس `import` میشه که ممکنه زمانبر باشه و برنامه رو کند کنه که بهینه نیست در حین `import` کردن شئ ساخته بشه.

راه حل اینه که یک تابع برای ایجاد شی تعریف کنیم و تابع رو بجای شئ `import` کنیم و بعد اینکه `import` کردن تموم شد هر جا خواستیم تابع رو کال میکنیم و شئ رو میسازه و برمیگردونه یا شی ساخته شده که از قبل وجود داره رو برمیگردونه که در این صورت در حین `import` کردن شی ساخته نمیشه بلکه بعد اینکه `import` کردن تموم شد و هر جایی تابع رو کال کردیم شئ ساخته\ساخته شده رو ریترن میشه و هر چقدر همزمان بر باشه دیگه عمل `import` کردن معطل ساخت شی نمیشه

.C107

روش استفاده از یک فایل کد به عنوان `:module or script`

مثلاً میخوایم `script/module` ایی بنویسیم که امکان استفاده هم بعنوان `module` و هم بعنوان `script` را داشته باشد. یعنی اگر فایل رو بصورت `module` استفاده کردیم یسری کدها اجرا بشه و اگه بصورت `script` اجرا کردیم یه سری کدها اجرا بشه.

با استفاده از نام فایل در شرایط مختلف تشخیص میدیم که فایل مورد نظر `script` است یا `module` مثلاً فایلی بنام `test1.py` داریم در این صورت اگر فایل(`test1.py`) رو بصورت `script` ران کنیم

`(__name__ == '__main__')` ولی اگر بصورت `module` ران کنیم `(__name__ != '__main__')`

در حقیقت دستورات از هم تفکیک میشه و اصطلاحاً میگیم آغا اگر بصورت `script` ران شدی این دستورات رو اجرا کن! و اگه بصورت `module` ران شدی این دستورات رو اجرا کن.



```
test1.py ×
1  class Car:
2      pass
3
4
5      1 usage
6      def main():
7          pass
8
9      ▶  if __name__ == '__main__':
10         main()
```

ماژول ها را از کجا ها میشه import کرد؟

چند تا جا هست که وقتی یک ماژولی رو میخوایم import کنیم در هنگام import کردن از اون جاها import میشه و اگه اونجاها نبود خطأ میده:

- پیشفرض هایی که موقع نصب اجرا شده

- دایرکتوری جاری ایی که اسکریپت تو ش هست و برخی دایرکتوری های دیگر

لیستی از کل دایرکتوری هایی که مفسر بهشون دسترسی داره:

The screenshot shows the PyCharm IDE interface. At the top, there's a code editor window titled 'test.py' containing the following code:

```
1 import sys
2
3 for i in sys.path:
4     print(i)
5
```

Below the code editor is a terminal window titled 'Run' showing the output of the script:

```
for i in sys.path
```

The terminal output lists the current Python path:

```
C:\Users\arazp\AppData\Local\Programs\Python\Python312\python.exe C:\Users\arazp\OneD
C:\Users\arazp\OneDrive\Desktop\basic_10-2\python_9-25\projects\python_project2\venv
C:\Users\arazp\OneDrive\Desktop\basic_10-2\python_9-25\projects\python_project2
C:\Users\arazp\AppData\Local\Programs\Python\Python312\python312.zip
C:\Users\arazp\AppData\Local\Programs\Python\Python312\DLLs
C:\Users\arazp\AppData\Local\Programs\Python\Python312\Lib
C:\Users\arazp\AppData\Local\Programs\Python\Python312
C:\Users\arazp\AppData\Local\Programs\Python\Python312\Lib\site-packages
C:\Users\arazp\AppData\Local\Programs\Python\Python312\Lib\site-packages\win32
C:\Users\arazp\AppData\Local\Programs\Python\Python312\Lib\site-packages\win32\lib
C:\Users\arazp\AppData\Local\Programs\Python\Python312\Lib\site-packages\Pythonwin

Process finished with exit code 0
```

: بصورت دستی نیز میتونیم بهش میسر append کنیم تا به ماژول ها یا فایل های مورد نظر note دسترسی پیدا کنیم.

: module در مورد برخی نکات

ماژول ها با snake\_case نام گذاری میشوند. میتوانیم print(moduleName) کنیم تا مسیر و دیگر مشخصات مختص ماژول مورد نظر رو مشاهده کنیم.

پایتون خیلی رو ثابت ها تمرکزی نداره.

**Module documentation string:** we can write documentation string in top of module for module and in body of function for function and also we can access to they from other scripts.

test1.py

```
1 """documentation string my module"""
2
3 x = 2
4 y = 3
5
6
7 def pow2(num):
8     """
9         Gets a number and returns it ** 2.
10        :param num:
11        :return:
12    """
13
14    return num ** 2
```

test2.py

```
1 import test1
2
3 print(test1.__doc__, end='\n\n')
4 help(test1)
```

Run

```
C:\Users\arazp\AppData\Local\Programs\Python
documentation string my module

Help on module test1:

NAME
    test1 - documentation string my module

FUNCTIONS
    pow2(num)
        Gets a number and returns it ** 2.
        :param num:
        :return:

DATA
    x = 2
    y = 3

FILE
    c:\users\arazp\onedrive\desktop\basic_10

Process finished with exit code 0
```

## Names

`dir()` without input for access the current script names.

`moduleName.__name__`: get module name.

`moduleName.__file__`: path the file.

## Environment variables and Python versions

هنگام نصب پایتون وقتی تیک add to path میزنیم دقیقاً چه اتفاقی می‌افته؟ نزنیم چی میشه؟

توی سیستم یک قسمتی بنام enviroment variables هست که به دو بخش کلی User variables و System variables تقسیم میشه.

### User variables:

زدن add to path: هنگام نصب پایتون اگه user variables رو بزنیم به add to path دو تا directory اضافه میکنه یکی مسیر خود پایتون نصب شده هست و دیگری sub ایی از اون مسیر، ایی بنام .scripts

نzedن add to path: و اما اگه user variables رو نزنیم دیگه اون دوتا path به add to path نمیشه و default بالاترین نسخه نصب شده که آن در لیست user variables هست رو در نظر میگیره با اینکه شاید نسخه بالاتر از اون رو نصب کردیم ولی چون add to path اش رو نزدیم شناسایی نمیشه.

### System variables:

اما اگر به System variables مسیری(path) اضافه کنیم حتی ورژن پایتونش از ورژن پایتونی که اش در User variables اضافه شده پایین هم باشه بازم در هر صورت اولویت در نظر گرفتن با پایتون System variables هست مگر اینکه path هیچ پایتونی بهش اضافه نشده باشه در این صورت اولویت با User variables هست.

اگه چند نسخه پایتون رو همزمان رو سیستم داشته باشیم چی میشه؟  
اگه چند تا ورژن پایتون رو داشته باشیم default path اش از بالا اولین هست در نظر گرفته میشه.

و در ضمن میتونیم با move up و move down path هر دو اضافه شده پایتون رو default را تغیر بدیم.

### روش‌های add to path زدن پایتون:

اول: دوباره از اول نصب میکنیم و اینبار add to path میزنیم  
دوم: بصورت دستی path هارو به enviroment variables اضافه میکنیم.

## Python versions:

A.B.C

3.11.3

- زمانی افزایش پیدا میکنه که تغیرات خیلی بزرگ و محسوس باشه. (A)
- یه سری تغیرات مهم کوچیک که اعمال میشه. (B)
- زمان افزایش پیدا میکنه که یه سری باغها یا تغیرات کوچیک روی پروژه اعمال میشه. (C)

روش‌های اجرای نسخه‌های پایتون:

- **Running latest version of python:** for example, we have Python version 3.7, 3.8, 3.11; The following command will run the latest version, which here is 3.11.  
**command:** py -3
- **Running direct a specific version of python:**
  - a) **command:** py -3.12
  - b) **command:** C:\Users\arazp\AppData\Local\Programs\Python\Python312\python.exe

## Package manager

pip stands for package installer for Python

از pip برای پکیج هایی که توسط متخصصین دیگهای توسعه داده شده که به آنها پکیج های شخص ثالث نیز گفته میشند استفاده میشوند. Pip بصورت default روی خود پایتون هست.

pypi stands for Python packages index

یه سایتی که برای package هایی که منتشر میشوند در اینجا قرار میگیرند. و پکیج ها توسط pip از این سایت دریافت و در دایرکتوری site-packages نصب میشند.

### Pip commands:

pip install [PackageName]: Without a version number, it will install the latest version.

pip install [PackageName]==[PackageVersion]: installing a custom version.

pip install --upgrade/-U [PackageName]: updating a package.

pip list: list of installed packages.

pip list --outdated: show packages that need to be updated.

pip show [PackageName]: shows package details.

### Installing project requirement Packages on other system:

فرضًا میخوایم یه پروژه ای رو از یه سیستم ببریم روی یه سیستم دیگهای نصب و اجرا کنیم و معمولاً هر پروژه ای به یسری packages هایی برای اجرا بعنوان پیشنهاد، نیاز داره و باید packages هایی که نیاز داره رو باید رو سیستم جدید نصب کنیم برای اینکار اسم package هارو با اجرای دستور

pip list > [fileName.format]

داخل فایل مورد نظر میریزیم و بعد با دستور

pip install -r [fileName.format]

همه اون پکیج هارو در سیستم جدید نصب میکنیم.  
راه حل بهتر استفاده از محیط مجازی هستش.

## Virtual environment

یک دایرکتوری مجزا شامل پایتون و همه‌ی ماثول‌های مورد نیاز مختص یک پروژه درش قرار می‌گیره. با ایجاد `venv` در واقع یک شبیه سازی از پایتونی که همراه با `modules and packages` های اش که رو سیستم نصب شده رو برای هر پروژه به همراه `modules and packages` های اختصاصیش ایجاد می‌کنیم.

Make Virtual environment:

(a) این روش برای زمانیه که `venv` به صورت `default` رو پایتون نصب نیست و باید اونو توسط `pip` نصب کنیم `venv` (old versions) نصب :`venv`

`pip install virtualenv`

ایجاد ماشین مجازی:

اول `cd` می‌کنیم به `directory` ایی که می‌خوایم تو ش `venv` ایجاد کنیم. سپس دستور ایجاد ماشین مجازی را مینویسم

`virtualenv [directoryName/directoryPath]`

ایجاد ماشین مجازی با نسخه خاصی از پایتون:

`virtualenv --python=[versionNumber] [directoryName/directoryPath]`

: هنگام نصب `venv` در `cmd`، محیط مجازی روی پایتون نصب خواهد شد که اون پایتون رو سیستم انتخاب شده و همچنین هنگام ایجاد `venv` نیز از نسخه پایتونی استفاده خواهد شد که `default` سیستم هست.

(b) از `python3.3` به بعد یک ماثولی بنام `venv` به صورت `default` کنار پایتون نصب میشه که ازش میتوانیم برای ایجاد محیط مجازی استفاده کنیم:

with last python version: `python -m venv [directoryName/directoryPath]`

with specific python version: `py -[versionNumber] -m venv [directoryName/directoryPath]`

## installing packages:

### فعال سازی محیط مجازی:

هنگام نصب default packages or modules روی pip، اینها توسط default یا پایتونی که روی سیستم نصب شده، نصب خواهد شد و برای اینکه روی venv مورد نظر نصب بشه باید فایل activate محیط مجازی مورد نظر را بصورت زیر فعال کنیم:

`.\basic_10-8\python_10-8\projects\python_project2\env3\Scripts\activate`

سپس دستور های نصب packages or module های مورد نظر را میزنیم و تا روی venv مورد نظر نصب بشه.

### غیرفعال سازی محیط مجازی:

`.\basic_10-8\python_10-8\projects\python_project2\env3\Scripts\deactivate`

## 9) File

در فرهنگ کامپیوتر، پرونده یا فایل (به انگلیسی: File) به کوچکترین واحد منطقی ذخیره‌سازی بر روی دیسک گفته می‌شود که کاربر یا برنامه‌ساز، قادر به مشاهده و دستکاری آن است. در یک تعریف کلی، فایل مجموعه‌ای است دارای یک نام (و معمولاً دارای یک ساختار درونی مشخص) از نمونه‌های مختلف یک یا چند نوع رکورд.

از فایل برای ذخیره سازی بلند مدت داده‌ها در قالب خاصی استفاده می‌شود.

### Type of files:

همه فایل‌ها، اطلاعات یا هر چیزی که روی هارد ذخیره شده همه توسط filesystem تبدیل به byte هایی می‌شون که هر byte متشکل از 7bit هست، که مقدار هر bit آن یا 0 یا 1 هستش. فایل‌هایی مثل عکس، فیلم، متن، کدهای سیستم عامل یا هر چیزی که به ذهنتون میرسه!

اما فایل‌ها به دو دسته اصلیه، فایل‌های binary و textual برای درک بهتر و رفع محدودیت شناختی انسان بصورت صوری تقسیم می‌شون که در اصل اینجوری نیست اما با تأکید دوباره فایل‌ها بازم در نهایت به 0 یا 1 تبدیل می‌شون.

و اگر فایلی با encoding and decoding standard decode نشه به یه چیز نامفهومی تبدیل می‌شون.

encoding and decoding standard • Bibary : حالا مثلاً اگر فایلی که از نوع jpg هست رو اگر با standard decode مخصوص خودش بشه به همون عکس یا فایل مورد انتظار تبدیل می‌شون که روزانه هزاران بار ازشون استفاده می‌کنیم.

ولی اگر با encoding and decoding standard decode نشه و با استاندارد متنه decode بشه، مقدار هر بایتی که با کد هر یک از کاراکترهای هر زبانی ممکنه توش پیدا بشه.

در حالی که شاید هر بایتی که یک byte از عکس هست و با استاندارد متنه به متن تبدیل شده اگر با استاندارد خود عکس decode می‌شد ممکن بود مثلاً یه بایتی نشان دهنده رنگ یا صدا و یا اگه موسیقی بود یک نوتی و ... باشه که در استاندارد متنه تبدیل شده به کاراکتر!.

encoding and decoding standard • Textual : و همچنین فایل‌های متنه فایل‌هایی هستند که اگر با یک standard decode متنه به متن با معنی می‌شون. چون استاندارد متنه استانداردی قرارده مخصوص فایل‌های متنه هست.

## Open function

`open(fileNameOrPath, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)`

`mode='r'`

r	Open for reading (default)- Opening as textual- The pointer is placed at the beginning of the file- If the file does not exist, error occurs	rt	rb	
w	open for writing, its previous content is delete first- The pointer is placed at the beginning of the file- If the file does not exist, it will be created first and then opened	wt	wb	
a	open for writing, appending to the end of file if it exists- The pointer is placed at the end of the file- If the file does not exist, it will be created first and then opened- seek method does not work or it is disable	at	ab	
x	open for exclusive creation, failing if the file already exists- writing- The pointer is placed at the beginning of the file	xt	xb	
+	open for updating (reading and writing)- The pointer is placed at the end of the file	rt+	rb+	

r+	For writing and reading- if file does not exist occur error.  Note: وقتی مثلاً فایل در mode خواندن باز شده و ما میایم توش بنویسم، چون تو حالت درست باز نشده خطأ میده ولی حالا ما میخوایم اگر اشتباه هم باز شده دیگه خطأ نده و کارمون نصف نیمه بمونه و حداقل با چاپ یک massage به اجرای بقیه کدها برسه برای اینکار از + استفاده میکنیم.	rt+/r+t	rb+/r+b
w+	For writing and reading- its previous content is delete first- If the file does not exist, it will be created first and then opened.	wt+/w+t	wb+/w+b

a+	For writing and reading- The pointer is placed at the end of the file- its previous content is not deleted- appending to the end of file if it exists	at+/ a+t	ab+/ a+b
x+	For writing and creation- failing if the file already exists	xt+/ x+t	xb+/ x+b

**Note:** بعد هر عملیات، location ای بہ عملیاتی انجام داده قرار میگیرد.

t = text file

b = binary file

By default, all operations related to files are done in read(r) and textual(t)

مقدار mode از 3 قسمت تشکیل میشے قسمت اول هست و قسمت دوم نوع فایل در بازکردن، و قسمت سوم + هست.

اگر فایل binary رو با مود t باز کنیم مقدار بایت هارو چاپ میکنه.

و اگه فایل textual رو بصورت binary باز کنیم، اون فایل رو با نوع داده byte تبدیل میکنه و با جزئیات بیشتری نشون میده.

[1][2][3]

1=r,w,a,x, ...

2=t,b

3=+

Opening a file in the simplest way C81.

file methods:

<u>close()</u>	Closes the open file-  هر تغیری بعد اینکه فایل بسته شد روی فایل ثبت میشے و تا زمانی که بسته نشده روی internal buffer هستش مگر اینکه متده flush اجرا بشه و internal buffer رو خالی کنه و در os buffer رو خالی کنه بنویسه.
<u>detach()</u>	
<u>fileno()</u>	

<u>flush()</u>	Flushes the internal buffer هر موقع این متده را بشه internal buffer داخلی رو خالی میکنه و در os buffer بنویسه. و اگر بخوایم از os buffer هم در فایل بنویسیم باید os.fsync(f) را فراخوانی کنیم .C85
<u>isatty()</u>	
<u>write()</u>	Writes the specified string to the file- return number of bytes or characters اگر یک رشته رو که bytes هست رو داخل فایل binary بنویسم تعداد byte های کاراکتر return میشه. ولی اگر همان رشته یک رشته ساده باشد رو داخل یک فایل textual بنویسیم در این صورت تعداد کاراکتر return میشه .C84
<u>writelines()</u>	Writes every iterable to the file
<u>read(n)</u>	Returns the file content
<u>n</u>	مشخص کردن تعداد byte هایی که از اول تا آخر فایل باید بخونه- اگر بایتی کاراکتر داشته باشه بجای مقدار byte خود کاراکتر رو چاپ میکنه.
<u>readline(n)</u>	Returns one line from the file
<u>n</u>	مشخص کردن تعداد byte هایی که از اول تا آخر سطر باید بخونه- اگر بایتی کاراکتر داشته باشه بجای مقدار byte خود کاراکتر رو چاپ میکنه. <b>Note:</b> اگر در یک فایل 4 خط داشته باشیم و مثلًاً خط شماره 1، اگر تعداد همه بايت هاش 10 تا باشه و یک readline بزنیم بگیم، 3 بايت اول از اون خط رو بخون و میخونه اوکی؟! اوکی. ولی اگر یک readline دیگه بزنیم و بگیم 5 بايت دیگه بخون اینبار در readline بعدی نمیره از خط بعدی بخونه از همون مکانی که readline pointer در قبلی تا جایی که خونده همون جا مونده ادامه هر عملیاتی مثل خوندن یا هر چی رو میده. و یک نکته دیگه اینکه اگر در یک خطی تعداد بايت ها 10 تا باشه و یک readline بزنیم بگیم 15 بايت بخون دیگه از خطی بعدی کسری بايت هارو نمیخونه و فقط 10

	تا بایت رو میخونه
<a href="#"><u>readlines(n)</u></a>	Returns a list of lines from the file
<a href="#"><u>n</u></a>	<p>نسبت به قبليها متفاوت عمل ميکنه، يعني <math>n</math> رو به نزديکترین اندازه internal buffer گرد ميکنه</p> <p>مثلاً تو يك فاييل 3 خط داريم که مثلا خط اول 8 byte، خط دوم 12 byte، خط سوم 5 byte</p> <p>اگر بگيم 7، کل خط اول رو برميداره و يا مثلا بگيم 9 ، خط اول + خط دوم رو برميداره و يا اگر مثلا بگيم 19، بازم خط اول + خط دوم رو برميداره</p> <p>اما اگه بگيم 22، خط اول + خط دوم + خط سوم رو برميداره</p> <p>يا کل يك خط رو برميداره يا اصلاً برنميداره .C89</p>
<a href="#"><u>writable()</u></a>	Returns true or false to determine whether the file is writable.
<a href="#"><u>readable()</u></a>	Returns true or false to determine whether the file is readable.
<a href="#"><u>seek(n, m)</u></a>	changes the pointer position in a file and returns the new position.
<a href="#"><u>m</u></a>	<p>برای حالت binary اون عملیاتی که میخوای رو از کجا انجام بد؟</p> <p>= از ابتدا (default): در این حالت index مثبت داده میشه</p> <p>= موقعیت جاری: در این حالت index منفی و مثبت داده میشه</p> <p>= از انتهای: در این حالت index منفی داده میشه</p> <p>C90</p> <p>برای حالت textual اون عملیاتی که میخوای رو کجا انجام بد؟</p> <p>(توی فایل‌های textual یکی از argument ها باید 0 باشه)</p>
<a href="#"><u>seekable()</u></a>	
<a href="#"><u>tell()</u></a>	returns the pointer position(bytes number) in a file.
<a href="#"><u>truncate(n)</u></a>	<p>فايل رو به تعداد byte هايی که مشخص ميکنيم می بره</p> <p>يعني مثلاً تو يه فاييل که بايتها بشه 30، و مثلاً <math>n</math> رو بگيم 5 باشه، در اين صورت 5 بایت از اول فاييل رو نگه ميداره و بقیشو حذف ميکنه که در ساده‌ترین حالت شايد 5 بایب همون 5 کارکتر باشه.</p>

**Note:** اگر فایل تو حالت textual باز شده باشه منظور n تعداد byte هست ولی اگر تو حالت باز شده باشه منظورش تعداد کارکتر هست.

Changing the position of the pointer:

تغییر پوزیشن بیشتر با فایل‌های باینری اوکی تره.

**Note:** Text files created on DOS/Windows machines have different line endings than files created on Unix/Linux. DOS uses carriage return and line feed ("r\n") as a line ending, which Unix uses just line feed ("\n").

که این باعث میشه در DOS دو بایت به عنوان پایان خط اشغال بشه و Unix نیز 1 بایت به عنوان پایان خط اشغال بشه.

\r\n => 2 byte

\n => 1 byte

اگر فایل رو بصورت binary باز کنیم مشخصه که بین لاین‌ها از \r\n استفاده شده که 2 بایت اشغال میکنه و این دو تا کارکتر بک اسلش معمولاً جفت شده. در تغییر موقعیت pointer باید به این 2 بایت توجه کرد.

```
test2.py
1 f = open('text.txt', 'rb')    ↵ 3 ^ ^
2 print(f.read())
3
4 # result:
5 # b'reza\r\ndolati\r\nneda'
```

1	reza
2	dolati
3	neda

فهمیدن اینکه کدام کارکتر‌ها برای جدا کننده بین خطوط استفاده میشه.

encoding=None

: اگر کارکترهایی که در فایل نوشته یا خوانده میشے از range اسکی خارج بود، باید آرگومان `note` تعریف بشه تا متن به کارکتر های نامفهوم تفسیر نشه.

هر داده ای که میخوایم داخل یک فایل بنویسیم باید اون داده از نوع همون فایل باشد. مثلًاً یک رشته رو نمیتوانیم داخل فایلی بنویسیم که اون فایل از نوع `b` باز شده. باید به نوع همون تبدیل کنیم سپس داخلش بنویسم.

رشته ای(`b`) که دارای کارکترهای خارج از اسکی هست رو اگر بخوایم در فایل `binary` بنویسیم باید اون رشته رو با تابع `bytes(str, encoding=x)` تبدیل کنیم C83.

## buffering=-1

In computer science, a data buffer (or just internal buffer) is a region of a memory used to store data temporarily while it is being moved from one place to another. Typically, the data is stored in a buffer as it is retrieved from an input device (such as a microphone) or just before it is sent to an output device (such as speakers). However, a buffer may be used when data is moved between processes within a computer. That is comparable to buffers in telecommunication. buffers can be implemented in a fixed memory location in hardware or by using a virtual data buffer in software that points at a location in the physical memory.

In all cases, the data stored in a data buffer are stored on a physical storage medium. A majority of buffers are implemented in software, which typically use the faster RAM to store temporary data because of the much faster access time compared with hard disk drives. buffers are typically used when there is a difference between the rate at which data is received and the rate at which it can be processed, or in the case that these rates are variable, for example in a printer spooler or in online video streaming. In the distributed computing environment, data buffer is often implemented in the form of burst buffer, which provides distributed buffering service.

A internal buffer often adjusts timing by implementing a queue (or FIFO) algorithm in memory, simultaneously writing data into the queue at one rate and reading it at another rate.

حافظه میانگیر یا buffer (به انگلیسی: buffer)، در لغت به معنای حائل یا میانی است که در علوم مختلف به شکلی متفاوت به مفهوم یک حافظه میانجی یا موقت بکار می‌رود. عموماً در علوم کامپیوتر و الکترونیک اشاره به حافظه موقت یک سیستم دارد که داده‌های ورودی و خروجی رو در buffer، در صفت پخش، write، read، ... قرار میدهد.

داده‌های رد و بدل شده بین دو واحد مجزا را بصورت موقت نگه داری می‌کند.

Ex: مثلاً یه متنی رو میخوایم چاپ کنیم. در این مسلسل CPU سرعت انجام وظیفه خودش خیلی بالاتر از printer هست پس در نتیجه CPU نمیتونه صبر کنه یا آرام آرام کاراکترهارو بفرسته تا printer چاپ کنه

CPU

Printer

پس میاد کل متن پردازش شداس رو در حافظه buffer قرار میده و میره سراغ وظایف بعدی خودش و از این به بعد printer از buffer کارکترهارو میخونه و چاپ میکنه

CPU



Buffer



Printer

در واقع وظیفه buffer نگه داری وقت دادهها تا زمانی که نوبت پردازش اون داده برسه.  
پس buffer به کار ما سرعت میبخشه، به این صورت که دیگه یه device ایی  $x$  ایی دیگه معطل یه device دیگه نمی مونه.

بافر به دو سته کلی تقسیم میشه:

- **internal buffer**: بافری که مربوط به زبانهای برنامه نویسی یا مخصوص یک برنامه یا یک بخش خاصیه.
- **os buffer**: بافری که مربوط به بافر سیستم عامل هست و محتوای internal بافر اول به cpu یا هر مقصد دیگه داده میشه.

Buffering values:

0	بافر غیر فعال - مثلاً اگه بخوایم در فایل چیزی بنویسیم دیگه در بافر ذخیره نمیشه تا بعد به فایل منتقل بشه و مستقیماً در فایل نوشته میشه و ...	b
n num	هر عدد دلخواه - تا زمانی که به n نرسیده تو buffer نگه میداره و وقتی به n رسید بافر رو حالی میکنه و در فایل مینویسه .C88	b
-1	مقدار سایز ram رو میگیره به اندازه سایز buffer در buffer مینویسه و سپس به فایل منتقل میکنه. یعنی تا جایی که buffer پر نشده به فایل محتوای buffer رو منتقل نمیکنه در یافت سایز buffer: ک .C87	t
1	- خط به خط در buffer ذخیره میکنه و به فایل منتقل میکنه.. تا جایی که \n نخورده در buffer مینویسه و اگه یجایی \n خورد محتوای buffer رو به فایل منتقل میکنه و buffer رو خالی میکنه و تا زمانی که \n نخورده بازم در buffer مینویسه و سپس به فایل منتقل میکنه و این مراحل تکرار میشه. و استثناعن اگر هم یجایی تا آخر رشته \n نبود دیگه اون رشته رو تا آخر در buffer مینویسه و سپس به فایل منتقل میکنه درواقع تا جایی که یک خط در بافر کامل نشده در buffer مینویسه. حالا یجای با \n یک خط بودن مشخص میشه، یجایی میبینه دیگه متن به انتهای رسیده و یک خط مشخص میشه و ... .C86	t

### errors=None

اگر کارکترهایی تو متن باشه که به هر مشکلی اگر مواجه شد بجای خطای دیفالتی که میده، میتوانیم تنظیم کنیم که وقتی به اون مشکل خورد بجای خطای دیفالت چه رفتاری از خود نشون بده

strict	to raise a ValueError exception if there is an encoding error. The default value of None has the same effect.
ignore	ignores errors. Note that ignoring encoding errors can lead to data loss.
replace	causes a replacement marker (such as '?') to be inserted where there is malformed data.
surrogateescape	
xmlcharrefreplace	
backslashreplace	Converts character to \ux unicode
namereplace	Replaces character name instead of character

### newline=None

هر کدام از مقادیر زیر چیزی که در آخر هر عنصر لیست هست رو تایین میکنه که چی به عنوان آخرین کارکترها اونجا نوشته بشه C98.

None	'\n', '\r', or '\r\n' translates to \n
'' or '\n'	translates to operating system default that is in Unix '\n' and in windows is '\n\r'
\r\n	\r\n
\r	\r

## Context manager

این اشیاء شبیه دکوراتور هستند، یعنی قبل و بعد دستورات اصلی یه کاریابی انجام میدن و باز و بسته کردن یچیزی رو بر عهده دارند.

مثلاً یچیزی رو باز میکنه بعد اینکه کارما با اون تمام شد اونومی بند و منابع رو آزاد میکنه. مثلاً بستن اتصال شبکه، بستن اتصال به پایگاه داده، مدیریت log in multi threading رو بر عهده بگیره و... CM ها اشیایی هستند که یا از قبل وجود دارند و یا خودمون میسازیم شون. این اشیاء دارای متدهای `__enter__`, `__exit__` هستند.

ساختمان کلی ساخت CM و return شدهی متدهای `enter` در f قرار میگیره .C91

تمام CM ها با دستور with فعال میشن که اول متدهای `enter` رو صدا میزنند و در آخر متدهای `exit` رو صدا میزنند.

از CM ها در هنگام کار با فایلها استفاده میشه: بدترین حالت کارکردن با فایلها اینه که فایل رو بصورت ساده باز کنیم که در این صورت ممکنه در حین `read`, `write` ... کردن با فایلها ممکنه خطای رخ بده و دستورات تا انتهای اجرا نشه و فایل بسته نشه تغییرات ذخیره نشه.

برای این مشکل چند تا راه حل داریم:

- باز کردن فایل با دستور `try` و بستن فایل در بادی `finally`
- بازکردن فایل با دستور `with`

وقتی با تابع `open` یک شیعی ایجاد میکنیم قابلیت اینو داره که با `with` فعالش کنیم و وقتی با `with` شد دو تا متدهای `enter`, `exit` برآش صدا زده میشه.

توى فایلها متدهای `enter` وارد فایل میشه و انو برای نوشتن یا خواندن و ... باز میکنه و فایل رو `return` میکنه و سپس دستورات بدنی `with` اجرا میشه و در آخر با متدهای `exit` فایل بسته میشه .C92.

از دستور `with` بصورت تودرتو نیز میشه استفاده کرد ولی روش بیترش باز کردن فایل در یک خطه .C93

منابع همیشه محدود بوده و هست. هر منبعی که باز میشه تا استفاده بشه در پایان کار چه کار با موفقیت انجام بشه چه نشه باید اون منبع کلوز بشه برای کار با منابع باید دو مفهوم، setup و teardown رعایت بشه. Setup رو میشه وصل شدن به db و انجام کارها در نظر گرفت و دومی رو بستن کانکشن.

قبل ها این دو مفهوم با استفاده از try finally پیاده میشد ولی اکنون با with statement پیاده میشه که مثل همون یه منبعی رو اپن میکنه بعد هر عملی کلوز میکنه به این مدیریت و بازو بسته کردن فایل context manager گفته میشه.

بصورت اختصاصی هم میتوانیم context manager پیاده سازی کنیم. به این صورت که یک کلاس که دارای متدهای \_\_enter\_\_ و \_\_exit\_\_ هست رو کد نویسی میکنیم و متدهای setup و teardown را پیاده میکنه و متدهای exception را خریب رو.

یکی از پرکاربردترین statement های پایتون هست.

طبعیتاً ممکنه خطاهایی هم حین تعامل با منابع رخ بد که با استفاده از پارامترهای متدهای exit بسته به نوع خطأ واکنش مربوط به خطأ پیاده میشه.

:Parameters

exc\_type: کلاس خطأ(نوع خطأ)

exc\_val: پیغام خطأ.

exc\_tb: تریسیک

متدهای exit عز دیفالت false ریترن میکنه ولی اگر true ریترن کنیم هر استثنایی اگر رخ داد رو نادیده میگیره و مستقیم میره متدهای exit رو کال میکنه.

C93.1 ساخت context manager اختصاصی: در این cm شبیه سازیتابع open انجام شده

## Shey file standard

هر چیزی تو پایتون یک شئ هست، هر شئ ایی متدها و attribute های خودشو داره و فایل‌های متنی که باهاشون کار میکنیم و اون فایل رو به هر صورت که باز میکنیم اون فایل در نهایت یک شئ هست که متدهای مختلفی از جمله `read`, `write`, ... رو داره.

حالا ۳ تا شئ بصورت دیفالت تو پایتون وجود داره که بازم مثل اشیاء دیگه متدهای خودشو داره و کاری که انجام میده با کار با فایل‌ها مشابه هست!.

حالا این یعنی چی؟ یعنی در فایل‌های معمولی متدهای فایل روی یک فایلی با `name` and `format` مشخص اعمال میشه و عملیات مورد نظر خودش رو پیاده میکنه ولی در شئ فایل استاندارد بجای یک فایل با `name` and `format` مشخص متدهای فایل روی `keyboard`, `screen` اعمال میشه.

بصورت ساده یعنی در شی‌های فایل استاندارد اون منبعی که اطلاعات رو مینویسند بجای اینکه یک فایل باشه یک صفحه نمایش هست و بجای اینکه از فایل بخونند از یک کیبورد میخونه

شئ فایل استاندارد	بجای فایل	این‌ها از شئ فایل استاندارد استفاده میکنند
<code>stdin</code>	<code>keyboard</code>	<code>input()</code>
<code>stdout</code>	<code>screen</code>	<code>print()</code>
<code>stderr</code>	<code>screen</code>	<code>exception</code>

متدهای ستون ۳ در پشت صحنه از شئ فایل استاندارد استفاده میکنند  
مخاطب شئ فایل استاندارد بجای فایل، `x` های ستون دوم هست

پروسه‌ای که طی میشه به این صورته یعنی برای اینکه `input` ایی دریافت بشه از `stdin` استفاده میشه و همچنین برای اینکه `print` ایی انجام بشه از `stdout` استفاده میشه و برای اینکه یک `error` ایی رخ بده از `stderr` استفاده میشه C94.

تغییر مخاطب یا مقصد شی فایل استاندارد: عز عه دیفالت مخاطب شی فایل استاندارد, keyboard, screen هست ولی میتوانیم مخاطبش رو تغییر بدیم به فایل تا عملیاتش رو روی فایل انجام بدیم. به این صورت که اول شئ فایل استاندارد مورد نظر رو به یک متغیر اختصاصش میدیم تا بعداً اگه خواستیم به حالت اولیه برش گردونیم.

سپس مقدار شئ فایل استاندارد رو تغییر میدیم به یک فایل که با open باز میشه، سپس میتوانیم متدهاش رو روی اون فایل اجرا کنیم و در آخر مقدار شئ فایل استاندارد رو بهش برمیگردونیم تا ماهیت شی تغییر نکند C95.

اما

قسمت جالب ماجرا اینجاست که وقتی فقط sys را import میکنیم و با sys به شئ فایل مورد نظر دسترسی پیدا میکنیم و عملیات مورد نظر رو انجام میدیم در این صورت اگر تابع print رو صریحاً بنویسیم بجای print کردن در خروجی، در فایل print میکنه:، چون در کل script/module مقصود رو از خروجی به فایل تبدیل میکنه به همین خاطر خروجی تابع print در فایل ظاهر میشه.

در بالا نیز در رابطه با این مبحث که print, input در پشت صحنه از شئ های فایل استاندارد استفاده میکنند صحبت کردیم. و در اینجا همین مسله رخ میده.

اما اگر با استفاده از متغیر temp که قبل عملیات، شئ فایل استاندارد مورد نظر رو تو ش ذخیره کردیم، شئ فایل استاندارد رو به حالت قبلی برگردونیم به حالت عادی برمیگردد و اینبار نتیجه print در خروجی ظاهر میشه C96.

این توضیحات برای شئ فایل استانداردهای دیگر نیز صدق میکند C97.

## Print

`print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`

: فاصله دیفالت اشیاء در نتیجه چاپ این تابع، sep هست ولی میتوانیم با مقدار دهی sep این دیفالت رو تغییر بدیم.

: اینم دیفالت روی screen چاپ میکنه، ولی میتوانیم یه فایلی open کنیم و f رو بهش بدیم تا در فایل print کنه.

`f = open('text.txt', 'w')`

`print('test', file=f)`

: دیفالت false هست و تو با فرنگه میداره سپس به فایل منتقل میکنه.

## JSON

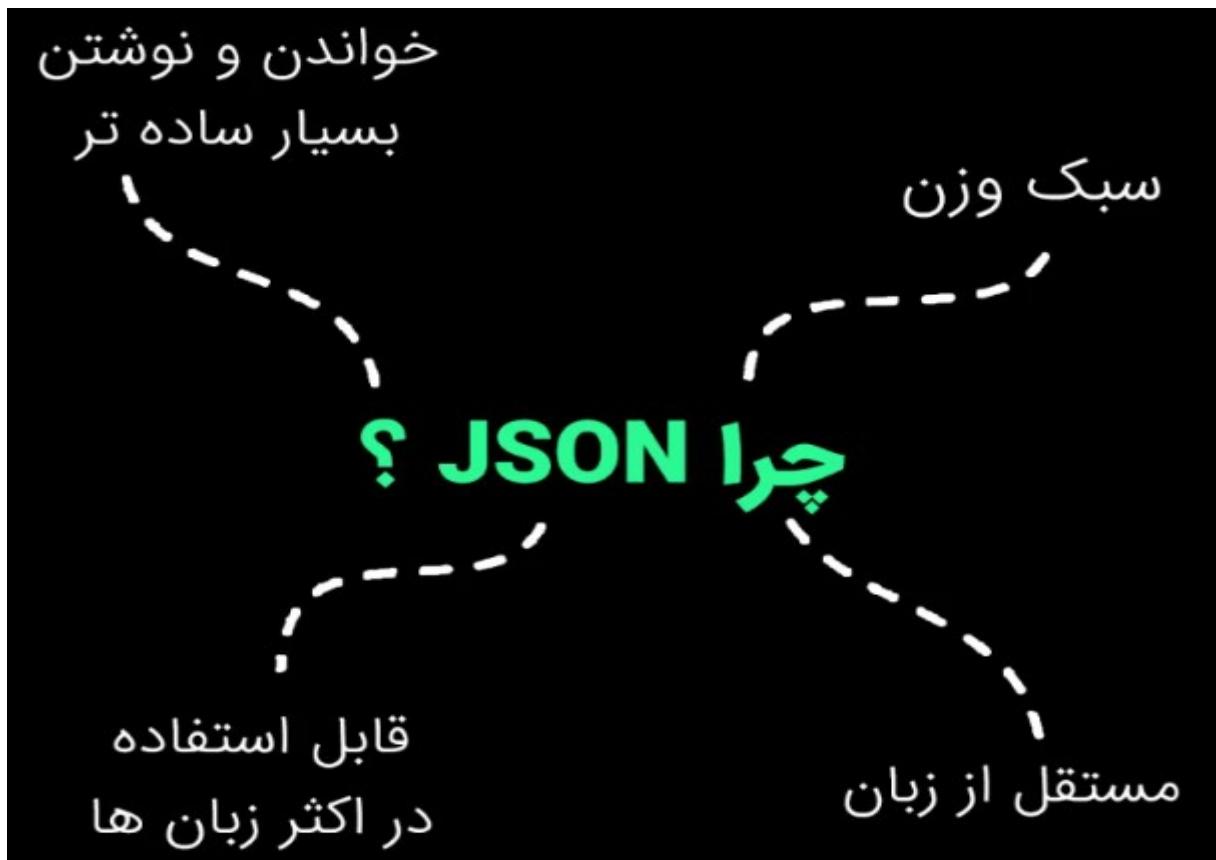
JSON is stands for JavaScript Object Notation and it is text, written with JavaScript object notation.

JSON is a syntax for storing and exchanging data. we use of JSON built-in package to work with json.

: جسون یک قالب استاندارد باز است که امکان تبادل داده‌ها در وب با استفاده از جفت‌های خصوصیت-کلید(key-value) را ممکن ساخته است.

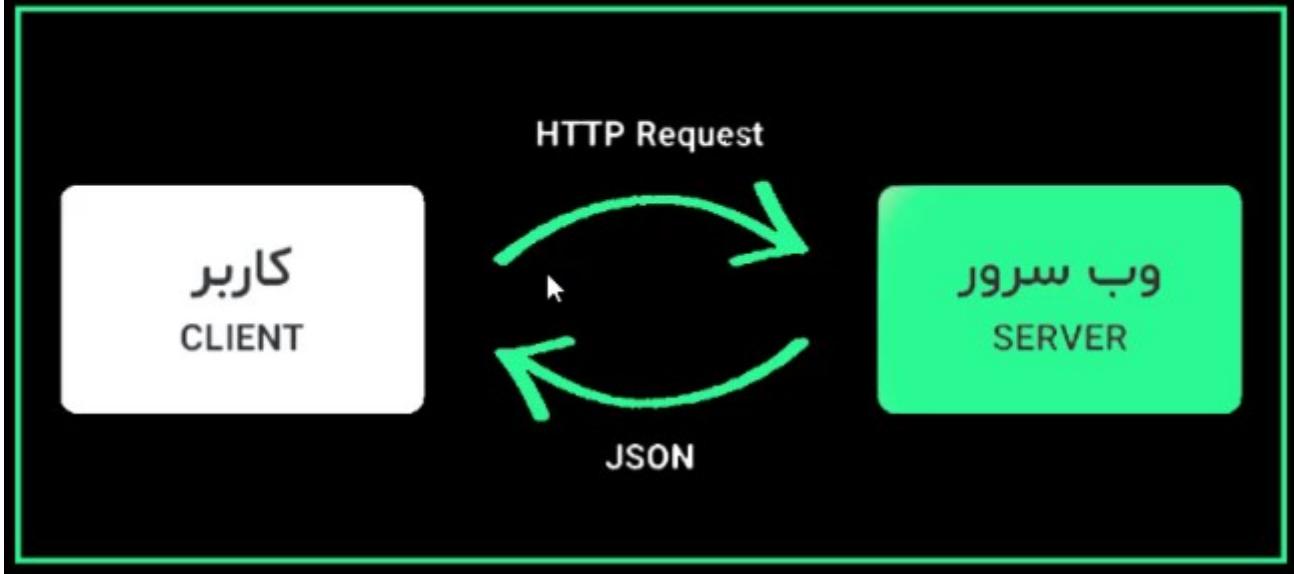
قراردادهای مورد استفاده‌ی جسون برای تمامی برنامه نویسان از جمله برنامه نویسان JS, Perl, Python, Java, Cpp ... شناخته شده است.

کاربرد JSON: یکی از دلایل اصلی استفاده از JSON اینه که، شکل اصلی داده‌هارو حفظ میکنه که از دلایل محبوبیت JSON هست. یعنی مثلًاً داده‌ای که int هست اگر تبدیل به فایل JSON بشه نوع خودش رو حفظ میکنه و int میمونه، درواقع ماهیت داده عوض نمیشه.



عمده کاربردش تو وب هست و اینطوریه که client یک request ایی send میکنه و server هم یه فایل JSON به عنوان نتیجه یا هر چی برمیگردونه.

# نحوه کار JSON



اشیاء JSON شبیه dictionary پایتون و آرایه ها نیز شبیه list ها در پایتون هست.

## قواعد نحوه نوشتن JSON

- + داده ها در جفت هایی بصورت نام / مقدار قرار می گیرد
- + داده ها با علامت کاما از هم جدا می شوند
- + علامت های {} اشیاء را نگه می دارند
- + علامت های [] آرایه ها را نگه می دارند

نمونه ای از JSON در JS:

# نمونه ای از جیسون

exemple.json

```
{  
    "age": 19,  
    "name": "Hema",  
    "isWebDeveloper": true,  
    "igUser": "@ItsHemaPie",  
    "hobbies": ["Coding", "BasketBall"]  
}
```

معادل نوع داده‌های int/float داده int/float پایتون، در JS تبدیل می‌شود به .Number

Python	JavaScript
dict	Object
list/tuple	array
str	string
int/float	Number
True/False	true, false
None	null

JSON methods:

load	Converts from JSON file to Python data types
loads(obj)	Converts from JSON string variable to Python data types
dump(obj, ...)	Converts from Python data type to JSON file
indent	To define the number of indents.
separators=(", ", ": ")	means using a comma and a space to separate each object, and a colon and a space to separate keys from values
sort_keys=True	To specify if the result should be sorted or not(default is False)
dumps(obj, ...)	Converts from Python data type to JSON string variable
indent	"
separators=(", ", ": ")	"
sort_keys=True	"

**Serialize/Parse:** converting from python data types to JSON file/string.

**Deserialize:** converting from JSON file/string to python data types.

## CSV

A simple way to store big data sets is to use csv files(comma separated values).

فایلی هست که معمولاً برای داده‌های ساختار یافته جدولی استفاده می‌شود. داده‌هایی که در اولین سطر نام خصوصیت هر ستون هست و در سطرهای بعدی در هر سطر یک موجودیت نوشته می‌شود.

CSV (Comma-Separated Values) is one of the prevalent and accessible file formats for storing and exchanging tabular data, such as a spreadsheet or database. A csv file stores tabular data (number and text) in plain text.

Each line of the file is a data record. Each record consists of one or more fields, separated commas.

### Reading a csv file:

- reader: returns an iterator object.

reading from csv file is done using the reader object. The csv file is opened as a csv file with python's built-in open() function, which returns a file object C99.

csvreader.line\_num: counter which returns the number of rows that have been iterated.

- DictReader: returns an iterator object.

reads each row of the CSV file as a dictionary where the keys are the column headers, and the values are the corresponding values in each row.

### Writing to a csv file:

- writer:

The file object is named as csvfile. The file object is converted to csv.writer object. We save the csv.writer object as csvwriter C100.

- csvwriter.writerow(fields): we use writerow method to write the row.
- csvwriter.writerows(rows): we use writerows method to write multiple rows at once.

## **Writing a dictionary to a csv file:**

To write a dictionary to a CSV file, the file object (csvfile) is converted to a DictWriter object **C101**.

- Writer.writeheader(): writing headers (field names) only in dic writing.

## 10) Class

oop

شی گرایی یعنی تمایل پیدا کردن برای مدلسازی اشیا.

شی گرایی یک مدل از زبان برنامه نویسی است که در آن بجای تعریف توابع و منطق از اشیاء و داده‌ها استفاده می‌شود. در این مدل از برنامه نویسی هر شی را می‌توان بعنوان یک مدل داده‌ای در نظر گرفت که دارای خصوصیات و ویژگی‌های منحصر به فرد می‌باشد. برای مثال اگر ماشین را یک شی در برنامه نویسی شی گرا تصور کنیم دارای خصوصیات منحصر به فردی مانند نام، رنگ، نوع، و ... است و همچنین دارای رفتارهای نیز می‌باشد مثل حرکت کردن، چرخش فرمان، عمل برف پاکن و ...

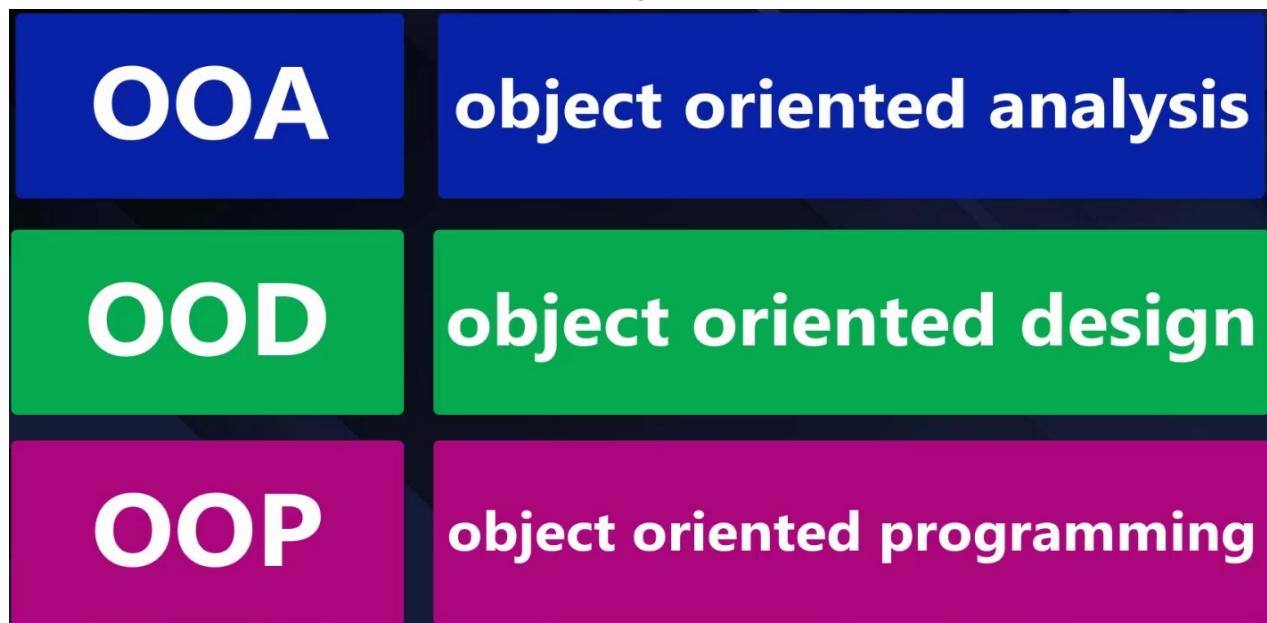


اشیای دیگری نیز می‌توان نام برد مثل همین ماوس دم دست شما، شی حساب بانکی، شی دانشجو

...

کلاس	رفتارها	صفات
حساب بانکی	واریز پول برداشت پول مشاهده موجودی	نام صاحب حساب شماره حساب میزان موجودی
دانشجو	محاسبه معدل انتخاب واحد	شماره دانشجویی تعداد واحد پاس شده رشته

پس شی به هر چیزی که یکسری خصوصیات و رفتارهایی داشته باشد اطلاق می‌شود.  
مراحل زیر مراحلی هستند که برای شی گرایی انجام می‌شود:



**Note:** البته در حال حاضر برای کل پروژه یکباره سه مرحله بالا رو انجام نمیدن بلکه پروژه رو به بخش‌های کوچیک تقسیم می‌کنند و برای هر بخش اون مراحل رو انجام میدن که باعث سهولت در کار می‌شه.

روشی برای برنامه نویسی هست که انواع مختلفی: Programming paradigms مثل functional programming, oop و ... داره که پایتون از هر دوی آن‌ها ساپورت می‌کنه. تو پایتون همه چیز شی هست و هر شیعی یا مستقیماً شی هست یا از ارتباط بین اشیاء ساخته می‌شه

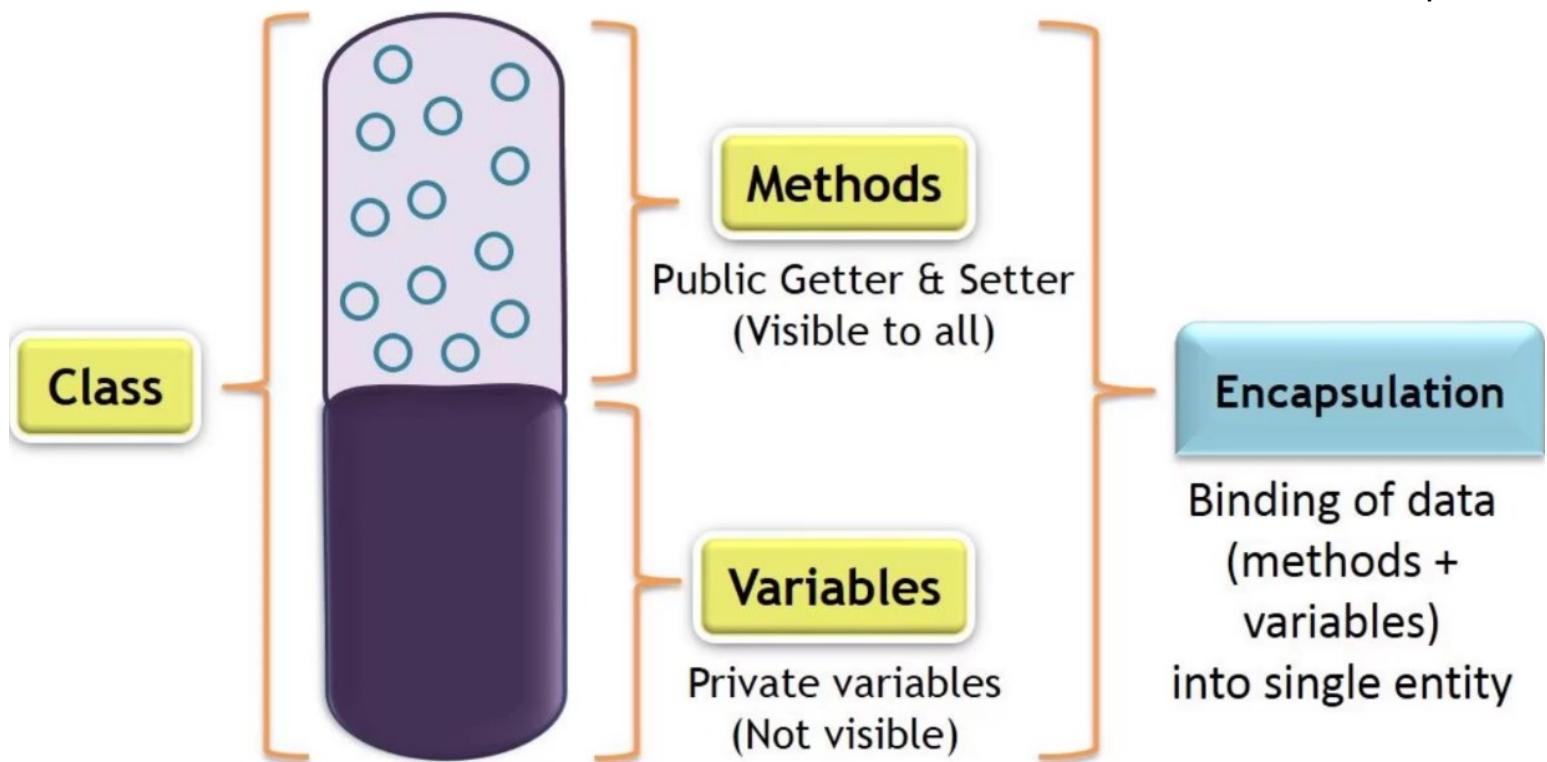
با استفاده از متدهای type می‌تونم نوع شی رو تشخیص بدیم.  
با id می‌توانیم به ایدی اون شی دسترسی پیدا کنیم.

## اصول شی گرایی Encapsulation (data hiding, interface)

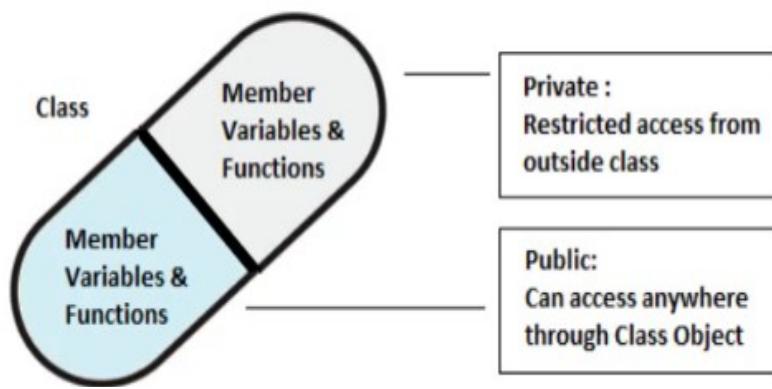
ویژگی‌ها مربوط به هر شی در محدوده و کلاس مربوط به خود شی قرار می‌گیرند. اشیاء دیگر قادر دسترسی و ایجاد تغییرات در داده‌های کلاس را ندارند و فقط می‌توانند به لیستی از توابع کلاس که بصورت عمومی تعریف شده‌اند دسترسی داشته باشند. این ویژگی در برنامه نویسی شی گرا باعث بالا رفتن امنیت و جلوگیری از فساد ناخواسته اطلاعات شده است.

کپسوله‌سازی بدین معنا است که تمامی اطلاعات مهم، درون شی نگهداری می‌شوند و نمی‌توان به آن‌ها از بیرون کلاس دسترسی داشت. به عبارتی، با ساخت یک شی به عنوان نمونه‌ای از کلاس، مشخصه‌ها و متدهای کپسوله‌سازی شده از بیرون کلاس قابل دسترسی و ویرایش نیستند و فقط می‌توان به داده‌هایی از کلاس دسترسی داشت که برنامه نویس آن‌ها را به صورت عمومی در اختیار سایر کاربران قرار داده است. این اصل شی گرایی باعث ایجاد امنیت بیشتر برای کدهای برنامه می‌شود.

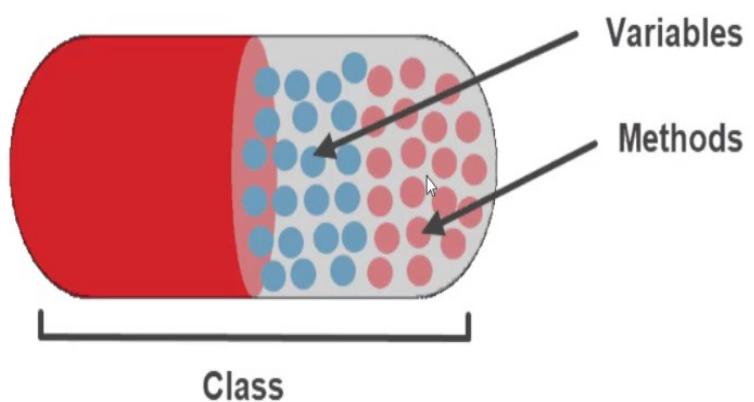
کپسوله سازی یعنی جمع کردن attributes و methods هایی که روی Class کار می‌کنند در کنار هم (در یک کلاس).



## Encapsulation and Data Hiding



## ENCAPSULATION



درواقع اگر در کنار هم جمع نکنیم و مثلاً تو یجا همه شماره حساب‌های یک بانک، یجایی همه موجودی‌ها، یجایی همه نام حساب‌ها و ...

در این صورت مثلاً یک لیستی از موجودی‌ها داریم و همچین لیست‌های دیگر که در این صورت باید از کجا معلوم میشند که هر موجودی مربوط به کدام شخص هست؟! یا هر شماره حساب مال چه کسیه؟! یا وقتی مثلاً یک واریز به حساب میخواهد انجام بشه از کجا معلوم میشه که به حساب چه کسی باید واریز بشه؟!

برای اینکار میایم کل چیزهای مربوط به یک شی رو در کلاس مربوطه دسته بندی میکنیم و پیچیدگی رو از بین میبریم که به این عمل encapsulation گفته میشه.

بنابراین هر چیز مربوط به یک شی فقط در صدا زدن همون شی در دسترسه، چون مربوط به اون شی هست و از طرفی هم کل چیزهای (خصوصیات و متدها) مربوط به اون شی اگر در کنار هم جمع بشوند اون شی رو میسازند

مثلاً در مثال ماشین موتور و گیربکس و تایر و بدنه و ... در کنار هم جمع می‌شوند تا شی ماشین رو میسازند. البته ممکنه هر یک شعی، شی‌های زیر مجموعه هم داشته باشه مثلاً میتوان موتور و گیربکس و ... رو بصورت جداگانه هر کدام رو یک شی در نظر گرفت که با جمع شدن در کنار یکدیگر یک شی دیگری مثل ماشین رو میسازند.

حالا بعد اینکه عمل encapsulation انجام شد یسری چیزها باید از دید کاربر باید مخفی بشه یا دسترسی به آنها از کاربر سلب بشه تا امنیت داده‌ها حفظ بشه که به این کار مخفی سازی یا data hiding گفته میشه

مثلاً در حساب بانکی کاربر نباید امکان دستکاری موجودی حساب بانکی را داشته باشد

توی پسری از زبان‌ها مثل cpp، Java، سی هشتک و... با پسری کلمات کلیدی مثل data hiding عمل private, public, protected را انجام میدن که در پایتون - این کارهارو انجام میده

در cpp دیفالت همه چیز private هست به همین خاطر در همچین زبان‌هایی اگر همه چیز private تعریف بشه، میشه بجای اینکه بگیم encapsulation، گفت data hiding.

سطح دسترسی در :cpp

:private: فقط در داخل کلاس در دسترس هست (میتونیم با تعریف متدهای داخل کلاس ریترنش کنیم) :Protected: فقط در خود کلاس و کلاس‌هایی که ازش ارث بری کرده‌اند در دسترس هست. :Public: در همه جا در دسترس هست.

حالت خصوصی | «رابط داخلی» (Internal Interface): به متدها و مشخصه‌هایی که حالت خصوصی دارند، فقط می‌توان از طریق سایر متدهای درون کلاس دسترسی داشت.

حالت عمومی | «رابط خارجی» (External Interface): به متدها و مشخصه‌هایی که حالت عمومی دارند، می‌توان از بیرون کلاس نیز دسترسی داشت.

### Encapsulation in python

اما در پایتون هر دو طرف کپسول شفاف هست و بخش کدری وجود نداره. ولی می‌توانیم با فرضیات یا قراردادهایی در نظر بگیریم که بعضی از متدها و خصوصیات در بخش کدر هستند. و با اینکه فرض می‌کنیم در بخش کدر هستند ولی در صورت نیاز هم قابل دسترسی هستند ولی در زبان‌های دیگه این بخش در دسترس نیست.

یعنی سیاست پایتون به اینصورت هست که می‌گه با فرضیاتی در نظر می‌گیریم که بعضی از مشخصات خصوصی، یا در بخش کدر هستند ولی من (پایتون) مانع دسترسی کاربر به بخش خصوصی نمی‌شوم. چون کاربر خودش می‌توانه تشخیص بدی که اینا خصوصی هستند و نباید بهشون دسترسی داشت و اگر هم می‌خواهد بهشون دسترسی پیدا کنه حتماً دلیلی داره.

پایتون با - برای ما مشخص می‌کنه که پسری اسمی private, protected هست ولی کاربر اگر بخواهد می‌توانه بهشون دسترسی پیدا کنه و توصیه می‌شه که دسترسی پیدا نکنیم.

حالتهای مختلف استفاده از \_:

(1) 1 \_ به تنهایی: بجای اسم متغیری که اهمیت نداره و ممکن‌هه استفاده هم نشه.

\_protected

(2) 1 \_ قبل اسم متدها یا متغیر:

برای import کردن مستقیم یک اسم از مژول: اون اسم حتی protected هم باشے به بصورت زیر from a import \_x

میشه import کرد:

Import کردن خود مژول: فقط اسم‌هایی که عمومی هستند import میشه ولی هنگام نوشتن import a

کد protected ها پیشنهاد نمیشن ولی میشه ازشون تو کد استفاده کرد:

\*: فقط اسمی عمومی import خواهد شد. مگر اینکه در داخل متغیر \_all\_ اون اسم protected or private هم بنویسیم تا بشه.

(3) 1 \_ بعد از اسم متدها یا متغیر: برای استفاده از کلمات کلیدی پایتون لایک \_len\_

\_private: not recommended

(4) 2 \_ قبل از اسم متدها یا متغیر:

: وقتی که قبل از اسمی 2 تا \_ استفاده میکنیم بصورت خودکار اون اسم تغیر میکنه مثلا خصوصیت \_phone\_ در کلاس Person \_phone تغیر میکنه به Person \_phone و با همین نام هم میتوانیم بهش دسترسی داشته باشیم.

و اگر یک کلاسی مثل Student از کلاس Person ارث بری کنه و اگر همون اسم رو دوباره تو خودش تعریف کنه باز هم مثل کلاس پدرش اسم کلاس به اسم اون خصوصیت اضافه میشه و به صورت \_Student \_phone در میاد و همچنین با همین نام هم میتوانیم بهش دسترسی پیدا کنیم.

و ضمن اینکه در کلاس فرزند Person \_phone equal with \_Student \_phone است چون student phone یک در هر حال یکیه.

با اینکه عمل name mangling اسامی private را تغیر میده و بیرون از کلاس باید با اسم جدید بهش دسترسی پیدا کنیم. ولی هم با اسم قبلی و هم با اسم جدید در داخل کلاس میتوانم به اون خصوصیت دسترسی داشته باشیم.

یک مزیت دستکاری نام اینه که اگر در کلاس فرزند خصوصیتی همنام با خصوصیتی که در کلاس پدر هست دیفاین بشه با اضافه کردن نام کلاس به خصوصیت‌ها، از تداخل نام جلوگیری میکنه .C108

\_specialMethods\_

(5) 2 \_ قبل و بعد از اسم متدها یا متغیر:

به این متدها special methods, magic methods, dunder methods و ... گفته میشه که متدهایی هستند که برای یسری کارهای خاصی با اسمی خاصی رزرو شده‌اند؛ تا تداخل نامی با اسمی ایی که توسط کاربر دیفاین میشن نداشته باشند.



interface

برای تعامل با هر چیزی نیاز به رابط داریم در ساده‌ترین مثال میشه گفت کنترل رابط بین انسان و تلویزیون هست یا کابل برق رابط بین پریز و دیوایس هست یا دستگیره رابط بین انسان و در هست فلسفه رابط به این صورته که میگه آغا من میخوام یه رابطی بیلد کنم که یه کاری رو برای من انجام بده حالا من نیاز ندارم که بدونم رابط چطوری بیلد شده یا چطوری یه کاری رو انجام میده؟! من فقط کاری که ازش میخوام رو انجام بده.

و در مقابل، در حین بیلد کردن یک interface به این نکته توجه میکنم که آیا من چی میخوام یا این interface چه کاری باید انجام بده؟ که با توجه به پاسخ سؤال interface بیلد میشه.

## Abstraction

با خاصیت abstraction، در برنامه نویسی شئ گرا در بدنه اصلی کدی نوشته نمیشود و به همین دلیل نمیتوانیم مستقیماً از روی کلاس‌های انتزاعی اشیاء را بسازیم. مفهوم انتزاعی چون کد مشخصی ندارد به برنامه نویسان کمک میکند تا به راحتی بتوانند تغیرات و افزودنی‌های خود را در طول زمان انجام دهند. برای مثال اگر در برنامه‌ای متدهای چاپ داشته باشیم میتوانیم بدنه این متدها را کدنویسی نکنیم. سپس کلاس‌های فرزند از کلاس انتزاعی بسازیم و کدهای بدنه را در این کلاس‌های فرزند تعریف کنیم و به راحتی از آن‌ها شی بسازیم.

انتزاع یعنی توجه به کلیات بدون توجه زیاد به جزئیات.

وقتی میگیم خودرو یا ماشین، وسیله خاصی وجود نداره که اسمش ماشین باشه بلکه این کلمات یک مفهوم کلی یا انتزاعی برای انواع اون وسیله‌ها است.

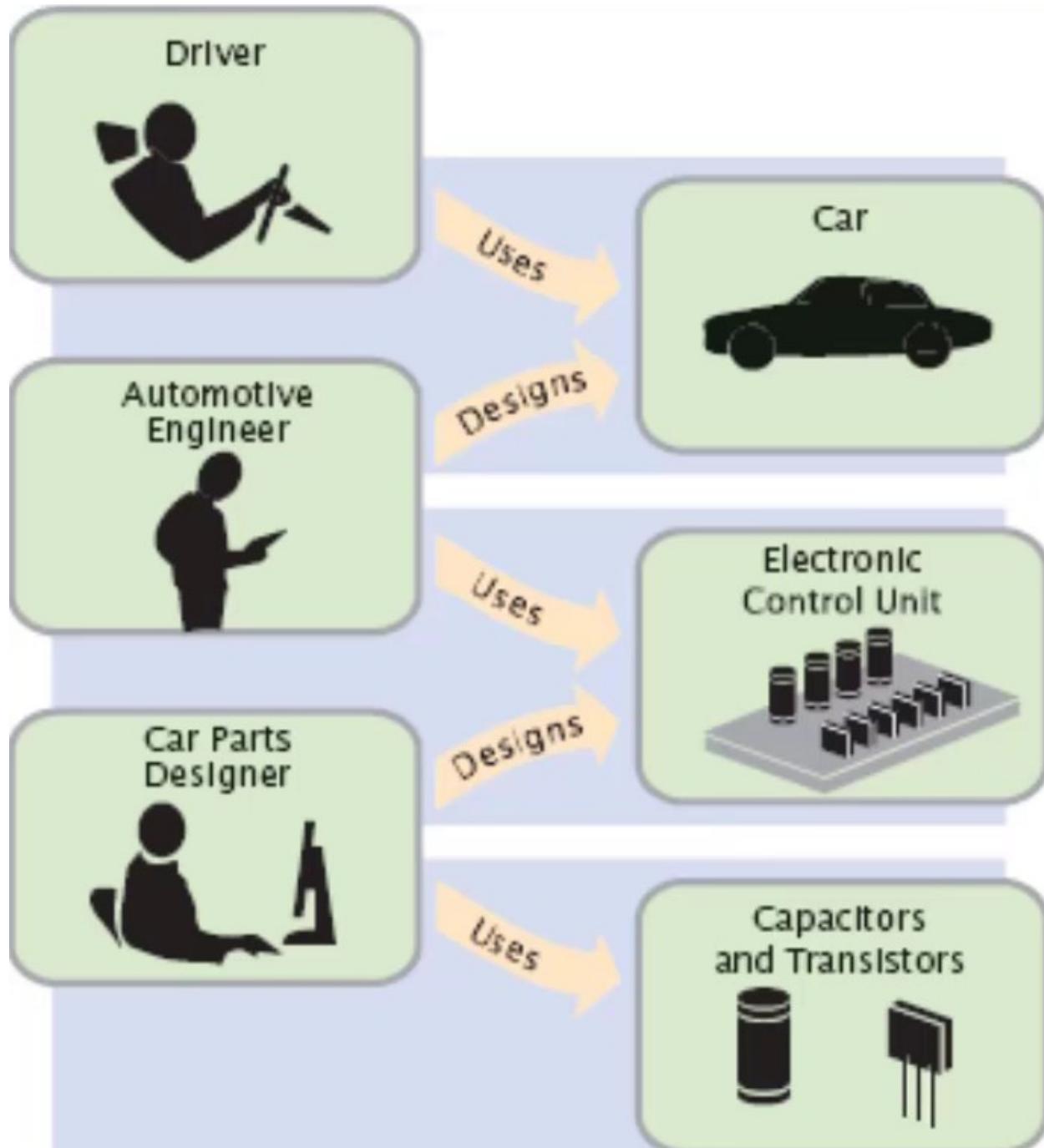
یا کلمات انتزاعی دیگری رو هم میشه از دنیای واقعی مثال زد مثل: حیوان، میوه، ورزش، غذا و ... هر ماشینی(پراید، لامبورگینی و...) یک نمونه واقعی/concrete ایی از اون کلاس هست مثلاً در انتزاع حیوان گربه یک concrete هست یا در انتزاع میوه، موز یک concrete هست



این کلمات انتزاعی در کل یکسری ویژگی‌های کلی مشترک نیز دارند. مثلاً ماشین، چرخ، فرمون، موتور، صندلی و ... رو در حالت کلی داره و میتونه در جزئیات یه ماشینی یک آپشنی رو داشته باشه که یه ماشین دیگه اون آپشن رو نداره یا بلعکس.

## Levels of abstraction

انتزاع در یک شی در سطح‌های مختلف میتوانه تفسیر بشه یا دید انتزاعی داشته باشه. مثلاً در شی ماشین انتزاع های زیر رو ممکنه وجود داشته باشه یا برای انتزاع مثلًا یک خوردنی سطوح انتزاع، خوردنی - میوه - موز - نوع موز. در نظر گرفت.



مثلاً در یک ماشین دید انتزاعی برای راننده در حد فهمیدن نحوه کار کردن با فرمان، دندن، پدال‌ها و ... هست و دیگه نیاز نیست که در سطح انتزاع پایین‌تری مثلاً بدونه که عملیات مکش، تراکم، انفجار، تخلیه در موتورچگونه یا کی اتفاق می‌فته. یا نیاز نیست بدونه گیربکس چگونه دندن هارو تعویض می‌کنه و ... .

اما برای یک تعویض کار ماشین سطح انتزاع عمیق‌تر یا پایین‌تر از سطح انتزاع راننده هست زیرا اون باید از همه پیاده‌سازی‌های داخلی اون ماشین سر در بیاره.

ولی در حالی که یک شخصی بعنوان تعمیرکار می‌تونه در سطح انتزاع پایین‌تر از تعویض کار هم قرار بگیره که اون قطعاتی که اون تعویض کار تعویض کرده رو باز کنه و اون قطعات رو در پایین‌ترین سطح انتزاع دید بزنه و تعمیر کنه

اما اصلاً می‌تونیم کسی که نمایشگاه ماشین داره رو بعنوان یه سطح بالاتر از همه سطوح‌های انتزاع تعمیرکار، تعویض کار، راننده که خیلی کلی‌تر به ماشین نگاه می‌کنه و دیگه کاری نداره که مثلاً دندن، فرمون، پدال‌ها چگونه کار می‌کنند و سطح انتزاعش در حدیه که مثلاً میدونه این ماشین پرایده اون یکی پژوه و ... که دیدگاه یا تعامل خیلی کلی‌ایی نسبت به اونها داره.

هر چه جزئیات کمتر سطح انتزاع بالاتر و بلعکس

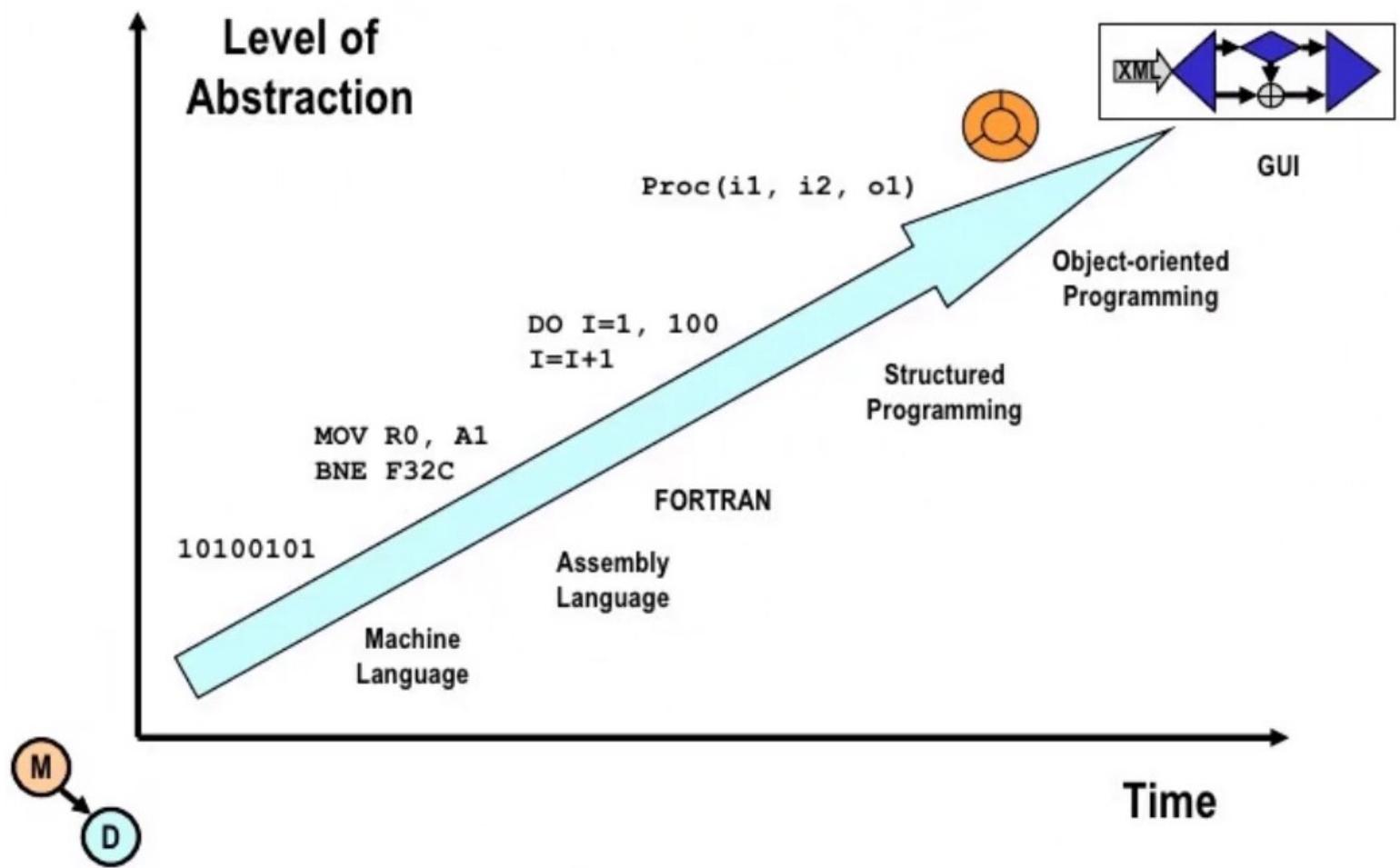
اصل انتزاع در شی گرایی چیست؟

سازندۀ دستگاه قهوه‌ساز به منظور طراحی و ساخت دستگاه، قبلًا به این اطلاعات فکر کرده است و تمامی این اطلاعات از دید کاربر این دستگاه پنهان است. کاربر دستگاه به منظور استفاده از آن، کافی است از دکمه‌های دستگاه استفاده کند و ورودی بدهد و خروجی تحويل بگیرد، بدون آن که لازم باشد از عملکرد داخلی دستگاه اطلاعاتی داشته باشد.

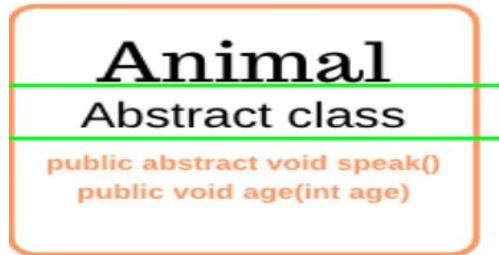
در شی گرایی نیز کافی است بدانیم از کدام متد شی می‌توان برای انجام وظیفه‌ای خاص استفاده کرد و چه مقداری باید به عنوان پارامتر ورودی این تابع در نظر گرفته شود. بدین‌ترتیب، دیگر نیاز نیست اطلاعاتی از نحوه پیاده‌سازی عملکرد تابع و انجام محاسبات آن به منظور بازگرداندن در اختیار داشته باشیم.

abstraction in computer science:

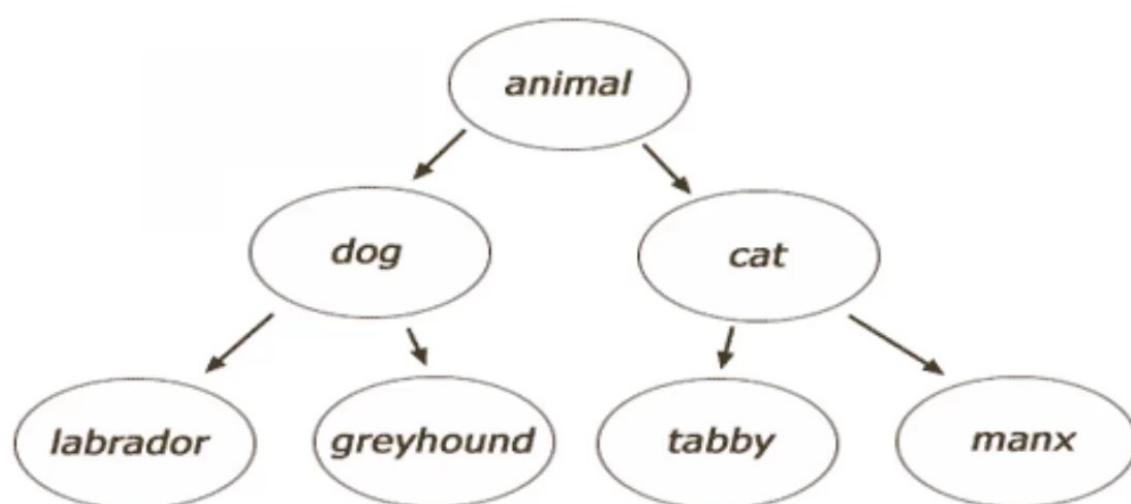
# Computer Science Is About Abstraction



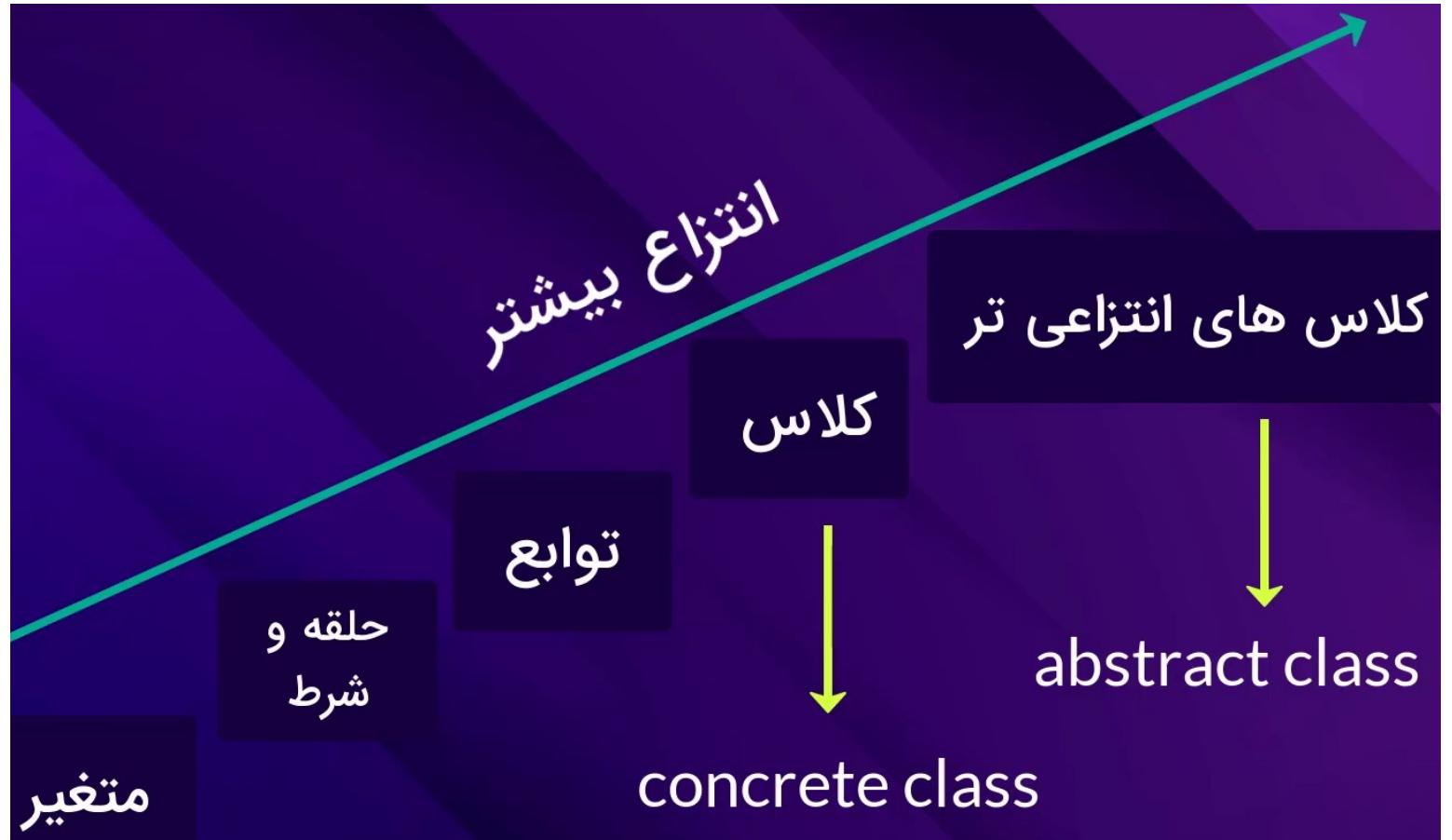
abstract class and concrete class:



## Inheritance



هر چه کلاس‌ها بالا تر میره کلاس‌ها انتزاعی تر هستند.



در پایین‌ترین سطح انزواع هر خط statement هست، یک لول بالاتر حلقه ها و شرط ها هست اگه بخوایم به حلقه ها و شرط ها انزواعی نگاه کنیم توابع رو در نظر می‌گیریم اگه بخوایم به توابع انزواعی نگاه کنیم کلاس هارو در نظر می‌گیریم و برای کلاس‌ها هم کلاس‌های abstract در نظر گرفته و پیاده می‌شوند هر چه انزواع بیشتر جزئیات کمتر



در تصویر بالا کلاس انتزاعی pet خصوصیات و ویژگی‌های مشترک را پیاده می‌کنند و کلاس‌های ساب ازش ارث بری می‌کنند و علاوه بر خصوصیات و ویژگی‌های ارث بری شده، خصوصیات و ویژگی‌های خاص خود را نیز پیاده می‌کنند.

## Abstract class

how to implement an abstract class: In this method, we inherit from the ABC class and use the abstractmethod decorator for abstract methods.

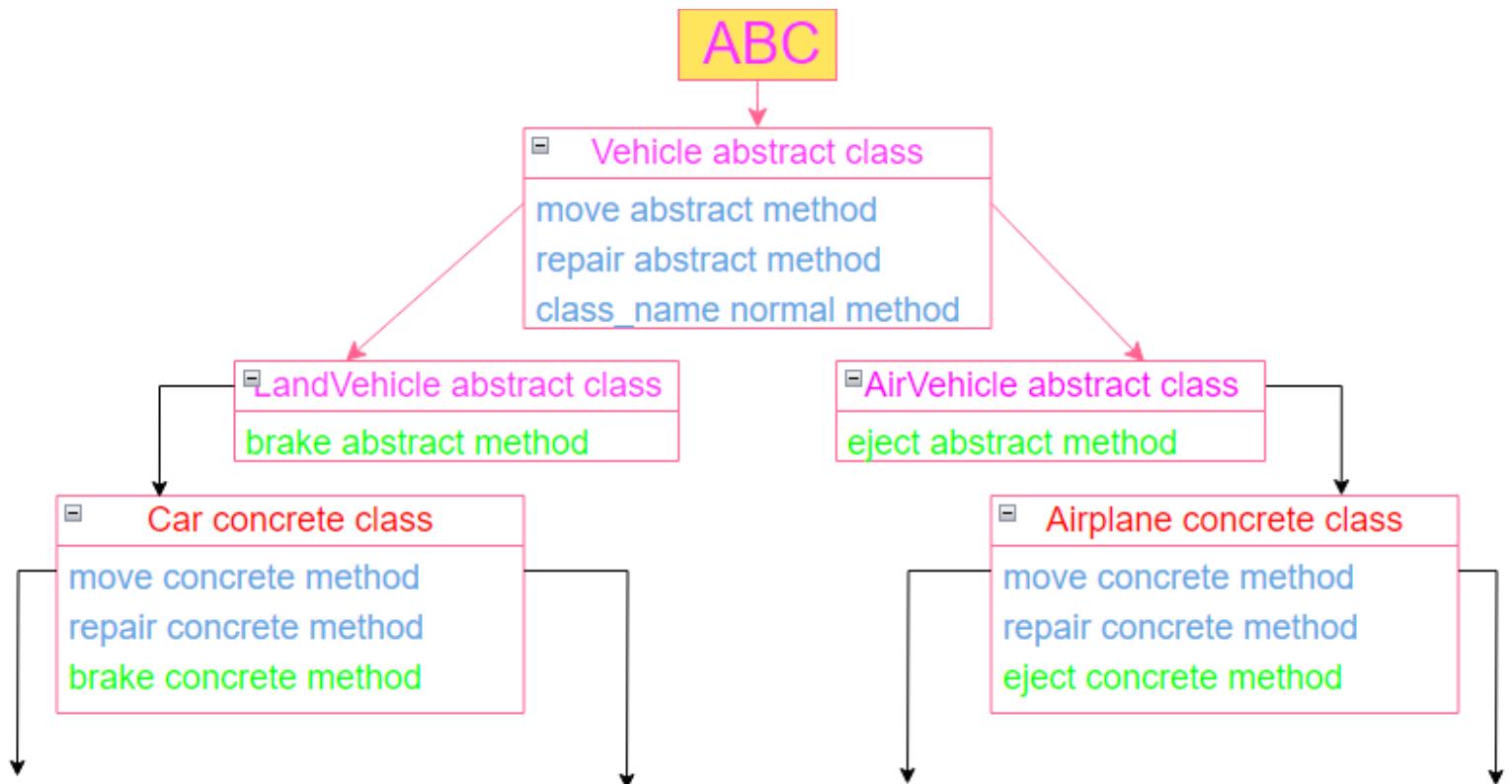
Abstract methods can have default implementations-C124.

ساب کلاس تا زمانی که متدهای انتزاعی کلاس پدر رو پیاده‌سازی نکرده، یک کلاس انتزاعی است.  
**Note**

Type of classes:

abstract

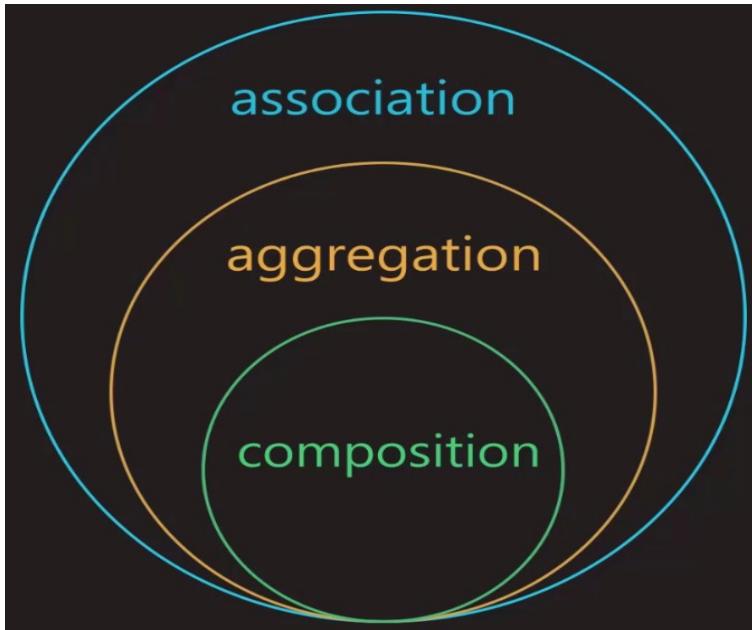
concrete



C125.

## The type of relationship between objects

در شی گرایی 4 نوع رابطه داریم، که یکیش از نوع `is_a` و 3 نوع دیگر از نوع `has_a` میباشد. اگر با توجه به این مفاهیم اشیا، موجودیت‌ها و روابط بین آنها را تعریف کنیم، شاکله‌ی! اصلی پروژه را بسته ایم.



هر رابطه `composition` هم یک رابطه `association` و هم یک رابطه `aggregation` هست.

و هر رابطه `aggregation` هم یک رابطه `association` هم هست

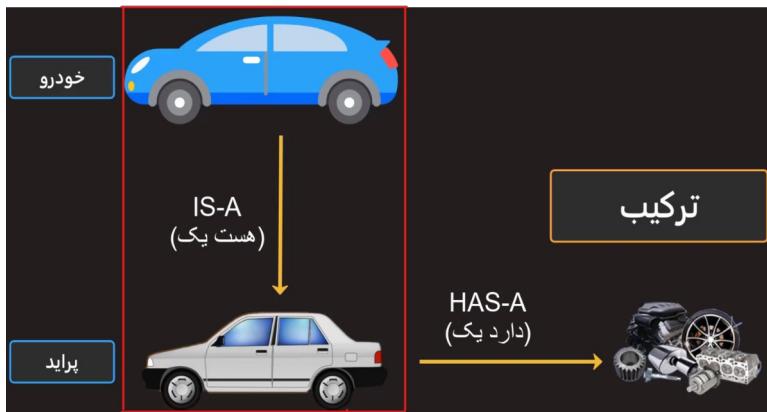
قبل هر پروژه تصمیم گرفته میشه که رابطه کدام اشیاء چی باشه. یعنی خود برنامه نویس تایین میکنه که روابط بین هر شیعی چه رابطه‌ای (`composition`, `aggregation`, `association`) باشه و در صورت از بین رفتن یک شیعی چه اشیایی باید از بین برن و ...  
مثالاً رابطه‌ی چرخ‌های ماشین‌های ایران خودرو با ماشین‌های تولیدیش معمولاً `aggregation` هست، چون چرخ‌هاش به هم میخوره.

ولی مثلاً رابطه یک ماشین خاصی با چرخ‌هاش ممکنه `composition` باشه. مثلاً رابطه بین لامبورگینی مدل X و چرخ‌هاش ممکنه `composition` باشه، چون ممکنه چرخ لامبورگینی به خودروی تویوتا نخوره! پس با نابودی لامبورگینی مدل X، چرخ‌هاش هم نابود میشه.

نوع رابطه یک شی یکسان با یک شی یکسان دیگر در یک پروژه با پروژه دیگه ممکنه متفاوت باشه.

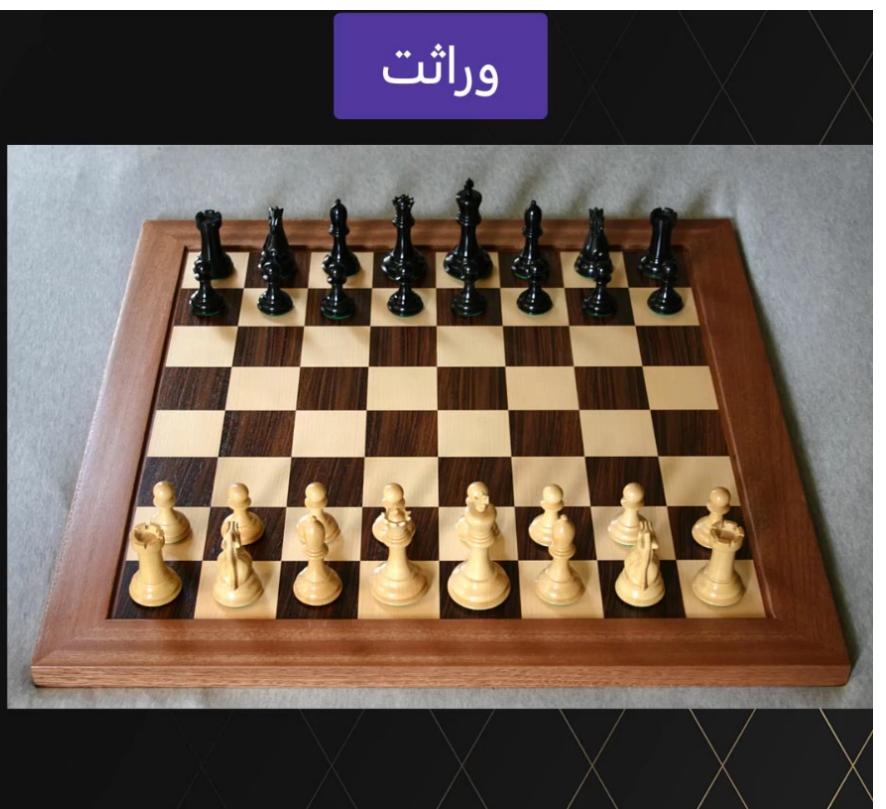
## Inheritance - is\_a

رابطه از جنس is\_a هست – مثلاً یک شیر یک حیوان است. یعنی صفات یه شئ دیگر را به ارث برد است.



رابطه کلاس انتزاعی خودرو با کلاس concrete پراید یک رابطه inheritance-is\_a هست. چون یک پراید یک خودرو هست.

در حالی که رابطه یک پراید با کلاس‌هایی مثلاً موتور، گیربکس، تایرها، و ... یک رابطه‌ی composition-has\_a هست. چون یک پراید (شئ پراید) دارای اون اشیاء موتور و ... هست.

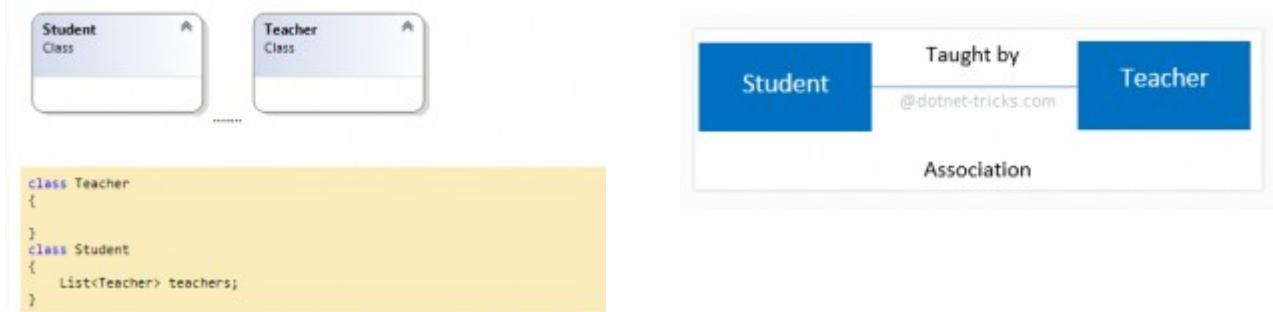


## Association/رابطه

رابطه‌ی بین اشیاء زمانی ایجاد می‌شود که یک شئ از شئ دیگر استفاده می‌کند. از لحاظ فنی، این رابطه مبتنی بر ارتباط بین دو کلاس است.

دو شئ با هم رابطه دارند اما وابستگی ضعیف است. رابطه محکم نیست و وابستگی شدید ندارد بالطبع تعلق و مالکیت وجود ندارد. هر شئی مستقله و کارهای خودشو می‌کنه ولی با ایجاد کانکشن بینشون کارهای دیگری هم می‌کنند که به یکدیگر مربوطه.

رابطه از بین بود در lifecycle اشیاء تغییری اینجاد نمی‌شود. و اشیاء مستقل از یکدیگرند. مثل رابطه‌ی دانشجو و استاد، راننده خودرو، کارمند بانک و حساب‌هایی که کاراوشونو می‌کنه.



يا رابطه راننده و ماشين.

يا رابطه در اينستاگرام. مثلاً در اينستاگرام اکانت a اکانت b رو فالو می‌کنه و يك رابطه association به وجود مياد. روابطشون سسته، يعني اين ارتباطشون اگه نباشه هم مشکل نيس. مثلاً اکانت a بدون فالو داشتن اکانت b هم ميتوشه به فعالیت خود ادامه بده و بلعکس.

و هر کدوم برای ادامه حیاط به دیگری وابسته نیستند. اگه دیگری باشه يا نباشه همچنان چرخه حیاط وجود داره.

**Note:** دقت کنید که در رابطه association يك شئ بخشی از شئ دیگه نیست.

## Aggregation – تجمع has\_a

aggregation شبیه به association است اما یک لول قوی‌تر از رابطه association دارد. مانند association، این رابطه زمانی که یک شئ از دیگر استفاده می‌کند توسعه می‌آید. در اینجا همه اشیاء چرخه عمر خود را دارند، اما مالکیت وجود دارد. این نشان دهنده رابطه "کل-بخش یا بخشی از" است. به عبارتی دیگر در این نوع رابطه هر شئ با اشیاء دیگر یک شئ جدید می‌سازند اما اون اشیایی که با ترکیب شدن یک شئ جدید می‌سازند به تنها یی هم یک شیعی هستند که کاربردی هستند و می‌توانند استفاده شوند lifecycle بخش (اشیا متشکل)، به lifecycle کل (شی اصلی) بستگی ندارد. شرط‌ها:

آیا می‌توان مستقل از هم باشند و استفاده شوند؟ بله  
و آیا نابود شدن یکی باعث نابود شدن اون یکی می‌شود؟ خیر



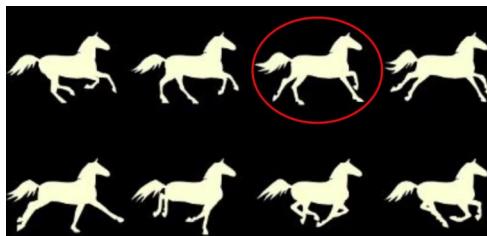
مثال 1: در زندگی واقعی، باتری تلفن همراه بخشی از تلفن همراه است. اگر به دلایلی (غیر از آسیب باتری) تلفن همراه غیرفعال شد، هنوز هم می‌توانیم از باتری در تلفن همراه دیگری استفاده کنیم. lifecycle بخش (باتری تلفن همراه)، به lifecycle کل (تلفن همراه) بستگی ندارد.

ما نمونه‌ای از CellBattery را خارج از کلاس CellPhone تعریف می‌کنیم. این تضمین می‌کند، در حالی که CellPhone تخریب می‌شود، CellBattery را می‌توان به برخی دیگر از CellPhone ها مجدداً استفاده کرد.

مثال 2:

هر اسپی جزئی از تصویر اصلی است ولی مستقل از تصویر اصلی هم معنی دارد و می‌تواند وجود داشته باشد.

ایجاد شدن و یا نابود شدن هر شئ(هر اسپ) به اشیاء دیگه(اسپهای دیگر) تأثیر نمیگذارد. به بیان ساده نابود شدن یک اسپ(یک شی) باعث نابودی یک اسپ دیگه(یک شی دیگه) نمیشه.



مثال3: چرخ های خودرو و خودرو  
تفاوت بین association و aggregation

هنگامی که اشیاء یک رابطه part-whole را ایجاد میکنند و طول عمر part وابسته به عمر whole نمیشود، رابطه به عنوان aggregation میباشد. در حالی که در part-whole association رابطه وجود ندارد.

در composition نیز، یک شئ از یک شئ دیگر استفاده میکند و یک رابطه part-whole ایجاد میکند. اما در اینجا، عمر part بستگی به چرخه حیاط whole دارد. این یک فرم تخصصی از aggregation است. این یک نوع قوی از aggregation است.

Ex: در این مثال دانشگاه بدون دانشجو باز هم دانشگاهه و دانشجو هم بدون دانشگاه یک شخص مستقله که زندگی و کارهای خودشو داره و نابودی یکی باعث نابودی اون یکی نمیشه .C121

## Composition/مرگ/ترکیب - has\_a

برای پیاده‌سازی انتزاع از دو تا مفهوم وراثت و ترکیب استفاده می‌شود.

یک لول قویتر از تجمع رابطه ترکیب با وابستگی‌های بیشتر نسبت به رابطه‌های قبلی هست.

ترکیب یعنی جمع آوری کردن چندتا شئ، برای ایجاد یک شئ جدید. برای زمانی استفاده می‌شود، که

یک شئ بخشی از شئ اصلی باشد در این رابطه شدت تعلق داشتن خیلی بیشتره.

در این رابطه اشیاء بشدت به هم وابسته هستند و بدون هم و یا مستقل از هم معنی ندارند و

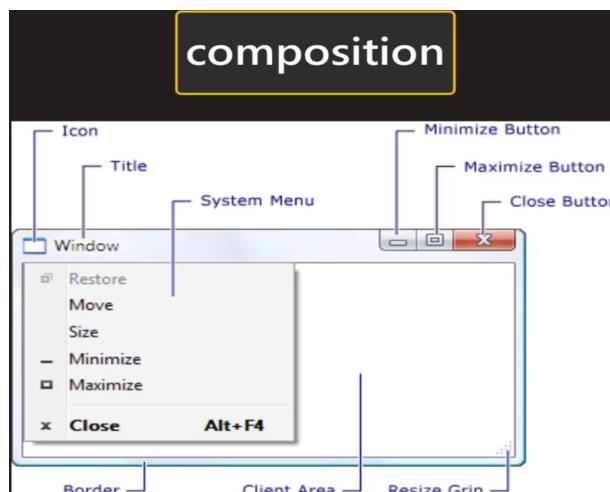
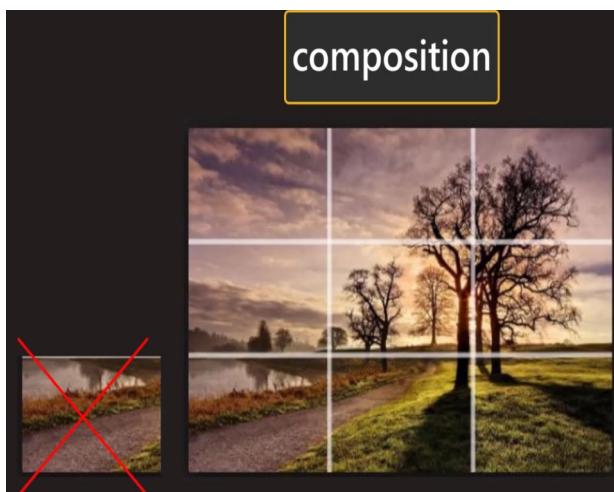
ممکنه کاربردی نباشند

شرطها:

آیا می‌توون مستقل از هم باشند و استفاده شوند؟ خیر

و آیا نابود شدن یکی باعث نابود شدن اون یکی می‌شود؟ بله

. lifecycle بخش(اشیا متشکل)، به کل(شی اصلی) بستگی دارد.



```
class Hotel
{
    List<Room> rooms = new List<Room> ();
}

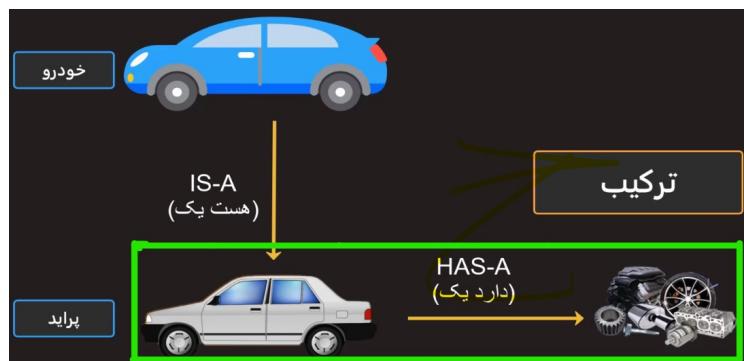
class Room
{}
```

در زندگی واقعی، اتاق‌ها بخشی از هتل هستند. اگر هتل نابود شود، قطعاً اتاق‌ها هم همان سرنوشت را خواهند داشت.

اتاقها به lifecycle هتل بستگی دارد. این سناریو زندگی واقعی از طریق برنامه نویسی بالا گرفته شده است. ما اتاقها را در داخل کلاس هتل ایجاد میکنیم. این نوع پیاده‌سازی تضمین میکند، در حالیکه هتل تخریب شده، اتاق نیز نابود شده است.

مثال 1: برگه امتحان. اگر برگه امتحان نابود بشه اشیاء دیگه مثل سؤالات، پاسخها، نمره و ... هم نابود میشه

مثال 2: رابطه کلاس‌های درس و اتاق استادان بدون هم معنی ندارند و اگه یکی نابود بشه اون یکی هم نابود میشه و کاربردی نداره.



مثال 3: همان‌طور که در تصویر هم قابل مشاهده هست

رابطه از نوع composition-has\_a هست چون یک پراید(شئ پراید) دارای اشیاء موتور و ... هست. و درواقع با جمعآوری کردن اشیایی مثل موتور، گیربکس، تایرها و ... در کنار هم یک شئ جدیدی بنام شئ پراید به وجود میاد.

در این مثال اشیاء موتور و ... بخشی از شئ پراید به حساب میاد.

در رابطه‌ی composition اشیائی که با هم در ارتباطن بدون هم معنی نمیدن. یعنی با ترکیب اونها یک شی جدید به وجود میاره که معنی پیدا میکنه. مثلاً close or minimize button بدون اون صفحه اصلی و محتواش معنی میده و کاربردی هستش.

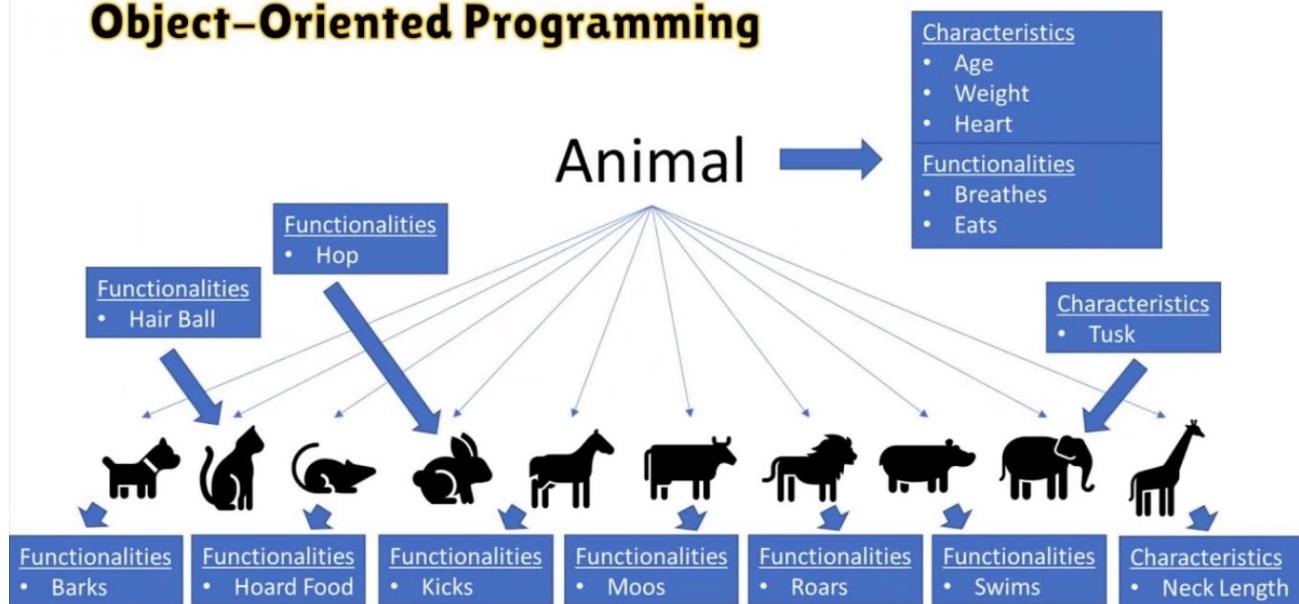
**Ex:** در این مثال سؤالات بدون برگه امتحانی معنی ندارند و برگه نابود بشه سؤالات هم نابود میشه. به همین خاطر نمونه سازی از سؤالات داخل کلاس برگه انجام میشه.

**C122:** یک سؤال و یک لیستی از انسورز Attributes

## Inheritance

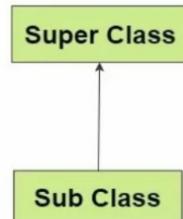
# Inheritance

## Object-Oriented Programming

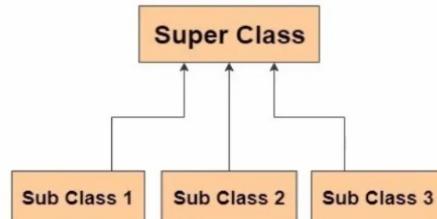


یک کلاس ویژگی‌ها و صفات یک کلاس یا چند کلاس دیگر را به ارث می‌برد و قادر است از آن‌ها استفاده کند. همچنین در کلاس فرزند می‌توان مشخصه‌ها و متدهای جدیدی را ایجاد کرد که در کلاس والد وجود ندارد و فقط مختص کلاس فرزند هستند.

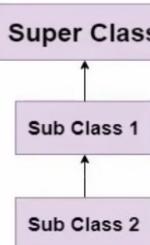
### Single Inheritance



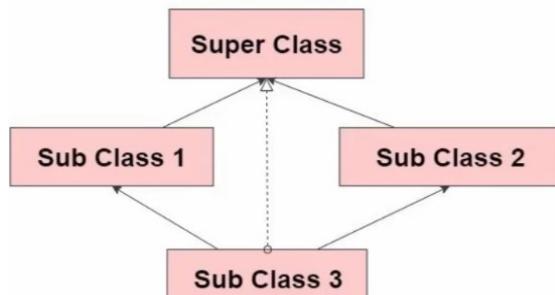
### Hierachial Inheritance



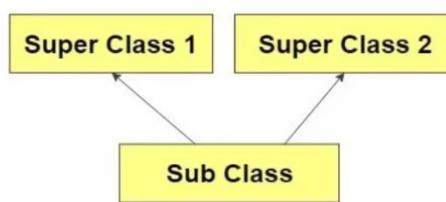
### MultiLevel Inheritance



### Hybrid Inheritance



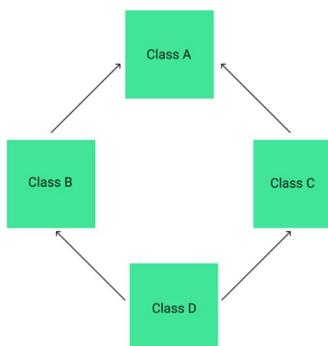
### Multiple Inheritance



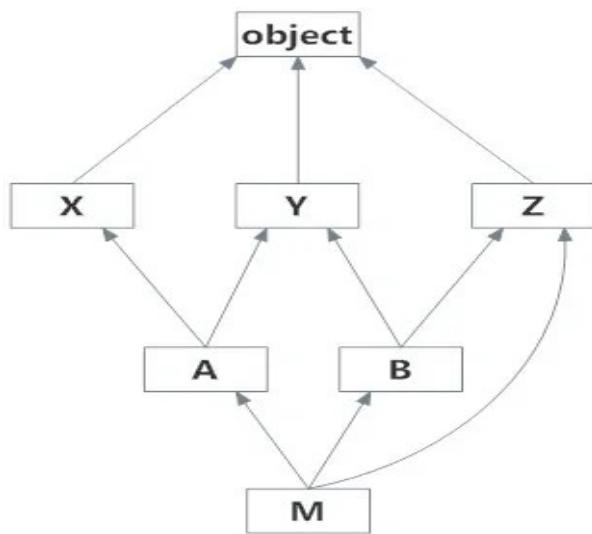
در این نوع وراثت، 1 کلاس فرزند تنها دارای 1 کلاس والد است. Single Inheritance: در این نوع وراثت، 1 کلاس فرزند دارای چند کلاس والد میباشد (multiple inheritance). با یک تابع super نمیتوانم متدهای init دو کلاس پدر را فراخوانی کنیم. راه حل استفاده از نام کلاس برای فراخوانی متدهای init هر کلاس و پاس کردن self و آرگومان مربوطه اش هست C111. initialization object in multiple inheritance:

وقتی از super استفاده میکنیم خیالمن راحته چون میدونیم از الگوریتم mro استفاده میکنیم. پس میتوانیم آرگومان هارو پاس کنیم. چون هر آرگومان به پارامتر مربوطه پاس میشه.

باید همه init های کلاس های پدرهاش رو باید فراخوانی کنه. پس آرگومان هارو بصورت که هر کلید دیکت رو به کلید همنامش تطبیق میده میفرستیم. با این کار طبق الگوریتم kwargs\*\* هر کلاس به نوبه خودش از داخل دیکشنری با کلید همنامش مقداردهی میشه C114.



از تابع سوپر استفاده میکنیم که از الگوریتم diamond problem solution برای رفع مشکل الماس استفاده میکنیم تا هیچ مشخصه ای دو بار کال نشه C113, C114.



مشخص کردن ترتیب جستجوی attribute ها و method ها هست. و اولویت از پایین به بالا و از چپ به راست هست.



همه کلاس ها در پایتون از کلاس object ارث بری میکنند. Note

M.mro(): to get mro order

MRO = M-A-X, B-Y-Z, object

**Multi-level**: در این نوع وراثت، کلاس فرزند از کلاس والدی ارث بری میکند که آن کلاس، بعنوان کلاس فرزند، از کلاس والد دیگری ارث میبرد.

**Hierarchical**: در این نوع وراثت، چند کلاس فرزند از 1 کلاس والد ارث بری میکند.  
**Hybrid**: این نوع وراثت، ترکیبی از روش‌های ارث بری منفرد و چندگانه و ... است.

`className._ _base_ _` : Get the names of inherited classes.

با ارث بری از کلاس‌های داخلی میتوانیم قابلیت [Inheritance from built-in classes](#) بهش اضافه کنیم. مثلاً یک قابلیت به کلاس `int`, `Dictionary`, ... اضافه کنیم.

[Super function/proxy object](#): این تابع برای استفاده از دستورات متدها و ویژگی‌های کلاس پدر در کلاس فرزند استفاده میشود. این تابع در مواقعي که میخواهیم علاوه بر استفاده از متدهای کلاس پدر، دستوراتی را نیز به آن متدهای اضافه کنیم، کاربرد دارد. این تابع از الگوریتم `mro` استفاده میکنه.

[seasons\\_10/packages/User.py](#)

## Mixin

کلاسی هست که فقط شامل متدهای مختلف با قابلیت‌های کاربردی درش پیاده شده‌اند که اگر کلاسی به این قابلیت‌ها نیاز داشت ازش ارث بری میکنه.

برخلاف کلاس‌های دیگر رابطه کلاس mixin با کلاس‌های فرزندش رابطه is-a نیست؛ چون هدف فقط افزون قابلیت به کلاس‌های فرزند هست نه پیاده‌سازی رابطه پدر و فرزندی.

مثلاً رابطه کلاس Cat با کلاس Animal یک رابطه is-a هست یعنی یک گربه یک حیوان هست و در مقابل که کلاس Cat که از کلاس میکس‌این، EatMixin ارث بری میکنه دیگه رابطه‌شون یک رابطه is-a نیست. چون یک گربه یک خوردن نیست! چون هدف فقط توسعه functionality کلاس‌های فرزند هست و چند تا تفاوت با کلاس‌های معمولی دارد: **C117**

Functionality  
هایی به کلاس‌های فرزند اضافه میکنه  
رابطه‌شون is-a نیست

انتهای اسمش یک Mixin اضافه میشه

از این کلاس‌ها object ایجاد نمیکنیم (چون هدف فقط توسعه functionality کلاس‌های فرزند هست و ازش ارث بری میشه)

چه زمانی از کلاس‌های mixin استفاده میکنیم: زمانی که میخوایم یک قابلیت اختیاری رو در کلاس‌های مورد نظر داشته باشم.

هر کلاس mixin یک قابلیت رو پیاده میکنه که در هر کلاس که به اون قابلیت نیاز شد اون کلاس از کلاس mixin ارث بری میکنه.

مثلاً ... هر کدام رو بصورت جدا در یک کلاس جدا پیاده میکنند تا فقط کلاس‌هایی که به اون قابلیت نیاز دارند اون قابلیت رو داشته باشند نه همه کلاس‌ها. چون مثلاً ممکنه کلاس میکس‌این، Music رو فقط برخی کلاس‌ها نیاز داشته باشند. مثلاً کلاس Car به کلاس میکس‌این، Music نیاز داره ولی کلاس MotorCycle به کلاس میکس‌این Music نیاز نداره. بخارتر اینکه کلاس‌هایی که به قابلیتی نیاز ندارند و اونهارو ارث بری میکنند و بخارتر اینکه ارث بری نکنند اون قابلیتهارو در کلاس پدر پیاده نمیکنیم **C118**.

## Different 'forms', same species



Light-morph jaguar

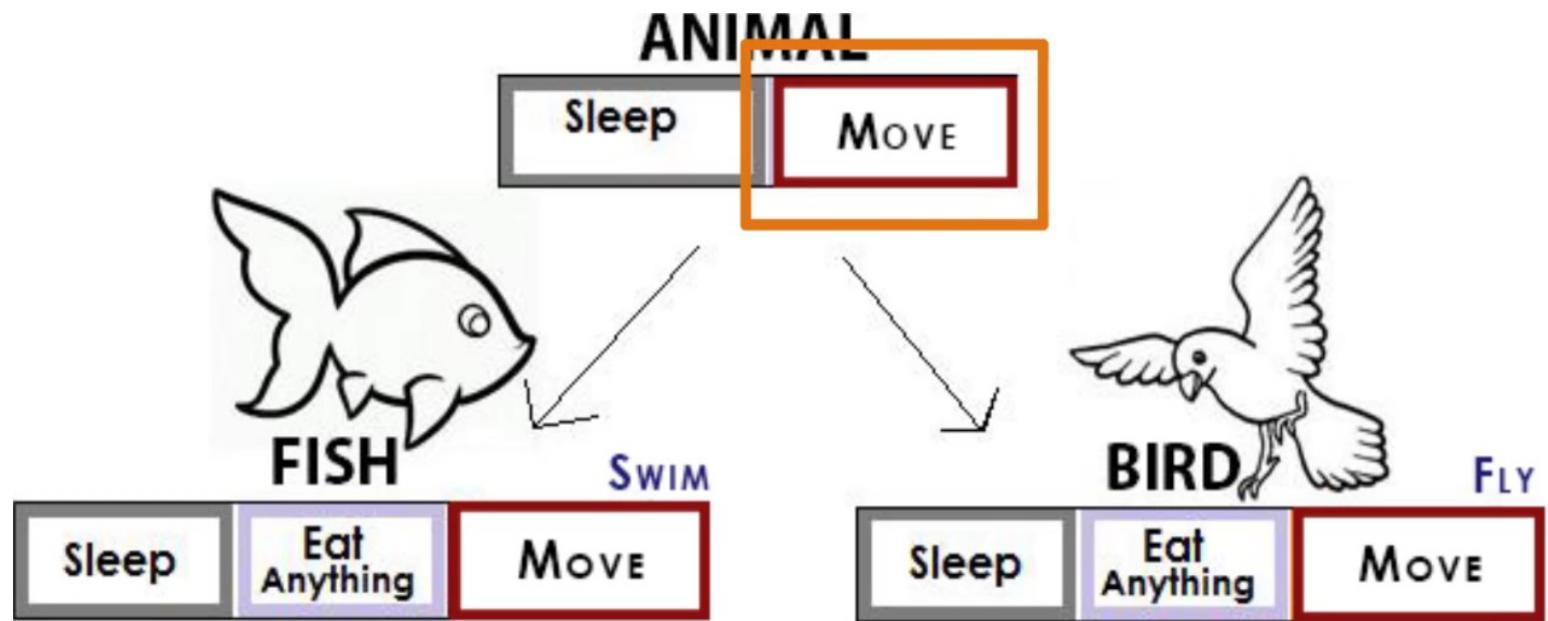


Dark-morph or black or melanistic jaguar

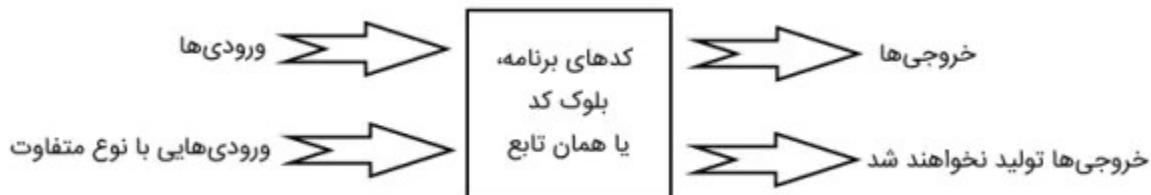
بر اساس اصل وراثت، کلاس فرزند میتواند متدها و مشخصه‌های کلاس والد را به ارث ببرد. با این حال، در برنامه‌نویسی شی گرا، اصلی با عنوان اصل چند ریختی وجود دارد که بر اساس آن، میتوان در کلاس فرزند، متدهای همانم با متدهای والد تعریف کرد، به طوری که عملکرد آن متده با عملکرد متدهای همانش در کلاس والد متفاوت باشد.

مثلاً در کلاس انتزاعی ماشین، متدهای انتزاعی حرکت تعریف میشوند که آن متده بعد از ارث بری در کلاس‌های ارث بری شده با بدنه‌های متفاوت برای هر وسیله نقلیه بصورت اختصاصی کدنویسی میشوند.

که به اینکه یک متده با نام یکسان در کلاس‌های مختلف با بدنه‌های مختص هر وسیله پیاده میشوند به همین خاطر به ظاهر شدن یک متده با نام یکسان و عملکرد متفاوت در کلاس‌های مختلف، به این عمل اصل چند ریختی گفته میشوند.



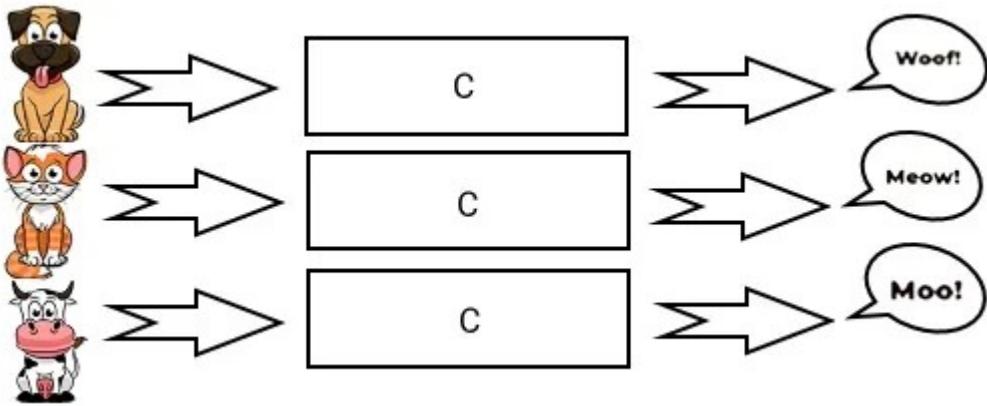
در واقع چندريختي يا پلي مورفيسم در برنامه نويسى روش متفاوتی از تفکر است. برای مثال، فرض می شود که چند خط کد يا بلوک از کدها وجود دارند. تعدادی ورودی به اين بلوک کد وارد می شوند و پس از اجرای کدها روی ورودیها، داده های جدید در خروجی تولید می شوند. یعنی اين بلوک کد که به نوعی می تواند همان تابع در برنامه نويسى باشد، عملیاتی را روی ورودیها اجرا می کند.



حالا اگر ورودی هایی با «نوع داده» متفاوت به این تابع وارد شوند، آیا این تابع باز هم خروجی تولید خواهد کرد؟ خیر. دلیلش این است که کدهای تابع انتظار دریافت نوع داده دیگری را ندارند و تنها می توانند عملیات مربوطه را روی نوع خاصی از داده ها مثلًا اعداد صحیح (Integer) انجام دهند. بنابراین، در این حالت، اجرای کدها با خطا مواجه خواهد شد. چندريختي يا پلي مورفيسم در برنامه نويسى برای حل این مشکل به وجود آمده است. حالا به بیان ساده پلي مورفيسم در برنامه نويسى چیست؟

پلي مورفيسم در واقع به این صورت است که چندین نسخه مختلف از همان کدها نوشته می شوند و هر نسخه از اين کدها يا توابع، ورودی هایی با نوع داده متفاوت را می پذیرند. مثلًا مطابق تصویر زیر، نسخه اول تنها نوع (شی) سگ از کلاس حیوان را به عنوان ورودی می پذیرد، نسخه دوم، نوع گربه و نسخه سوم هم تنها نوع گاو را در ورودی قبول می کند. نکته مهم این است که نامهای تمام این نسخه های کد یکسان هستند. برای مثال، نام همه بلوک کدها در تصویر زیر «C» است. همچنین، این

نسخه‌های متفاوت با نام یکسان، عملیات یکسانی را هم انجام می‌دهند و تنها نوع داده ورودی متفاوتی را می‌پذیرند. به این ترتیب، دیگر خطایی به وجود نخواهد آمد و تمام نسخه‌ها خروجی لازم را تولید خواهند کرد.



بنابراین، به بیان ساده، پلی مورفیسم در برنامه نویسی و شی گرایی، تولید یک بلوک گُد یکسان و مهم‌تر از همه، تعیین یک نام واحد و یکسان برای آن با این قابلیت است که آرگومان‌ها، متغیرها یا همان ورودی‌هایی با نوع داده متفاوت را می‌توان به آن وارد کرد. به واسطه قابلیت چندریختی، برنامه این امکان را خواهد داشت تا با مشخص شدن نوع داده ورودی، کد مناسب و مختص به آن نوع داده خاص را روی ورودی‌ها اجرا کند. مثلًاً اگر نوع داده یا شی ورودی سگ باشد، کدهای مخصوص نوع سگ اجرا خواهند شد.

یعنی مثلًاً اگر متّدی به نام `(() speak` وجود داشته باشد و ورودی از نوع گربه یا `Cat` باشد، متّد `(() speak` مربوط به گربه اجرا خواهد شد و اگر ورودی از نوع گاو باشد، متّد `(() speak` مخصوص گاو اجرا می‌شود. بنابراین پلی مورفیسم در برنامه نویسی به بیان ساده، استفاده از یک بلوک کد و تغییر «نسخه» آن بلوک کد، بر اساس نوع ورودی‌ها است.

اصطلاح چندریختی یا پلی مورفیسم در برنامه نویسی و علوم کامپیوتر به این معنا است که می‌توان به اشیایی با انواع مختلف از طریق رابط و واسطه یکسان دسترسی داشت. هر نوع می‌تواند پیاده‌سازی مستقل خود را از این واسطه فراهم کند.

چگونه می‌توان تعیین کرد یک شی پلی مورفیک است؟

برای اینکه بتوان مشخص کرد که آیا یک شی دارای حالت چندریختی یا به اصطلاح «پلی مورفیک» (Polymorphic) هست یا خیر، می‌توان یک آزمایش ساده انجام داد. در صورتی که آزمون‌های «`is-a`» (آیا هست؟) یا «`instanceof`» (نمونه‌ای از) برای آن شی موفقیت‌آمیز باشند، آن شی چندریختی دارد و به اصطلاح «پلی مورفیک» است.

: وقتی متدهایی با نام های یکسان و پارامترهای متفاوت داشته باشیم به این عمل **method overloading** می‌گویند.

**operator overloading**: می‌توان کارکرد یک «عملگر» (Operator) را بسته به «عمل وندهایی» (Operands) که استفاده کرده است تغییر داد. این کار با عنوان «سربارگذاری عملگرها» (operator overloading) نامیده می‌شود.

مثالاً عملگر + در جمع کردن int, float, string, complex هر کدام را به یک صورت جمع میکنے. به این صورته که مثلاً دو تا رشته را به هم وصل میکنے ولی دو تا عدد را باهم جمع میکنے و یک عدد جدید را به عنوان خروجی میده یا به اصطلاح فنی مثلاً اگر از عملگر - بین دو عدد استفاده کردیم از متدهای `sub` را از کلاس اینستنس مربوطه استفاده میکنیم یا اگر بین دو رشته قرار گرفت بازم از اون متدهای مربوطه از کلاس مربوطه استفاده میکنیم و ... .

مثالاً تو این مثال تعریف کردیم که هر موقع بین دو تا شی مشترک از این کلاس از عملگر < , - and ... استفاده شد سن هاشونو منها میکنیم و ریترن کنیم یا مقایسه بزرگتر بودن سنشونو میکنیم و ... . برای عملگرهای دیگه هم اینجوریه که دیفرنس بھیور در مورد هر تایپ از خود نشون میده .  
**C127**

## Binary Operators:

Operator	Magic Method
+	<code>__add__(self, other)</code>
-	<code>__sub__(self, other)</code>
*	<code>__mul__(self, other)</code>
/	<code>__truediv__(self, other)</code>
//	<code>__floordiv__(self, other)</code>
%	<code>__mod__(self, other)</code>
**	<code>__pow__(self, other)</code>
>>	<code>__rshift__(self, other)</code>
<<	<code>__lshift__(self, other)</code>
&	<code>__and__(self, other)</code>
	<code>__or__(self, other)</code>

<code>^</code>	<code>_xor_(self, other)</code>
----------------	---------------------------------

Comparison Operators:

<code>&lt;</code>	<code>_lt_(self, other)</code>
<code>&gt;</code>	<code>_gt_(self, other)</code>
<code>&lt;=</code>	<code>_le_(self, other)</code>

...

method overriding: بازنویسی کردن بدنه‌ی متدهای کلاس والد در کلاس‌های فرزند را overriding گویند.

type of polymorphism:

- subtype (runtime):

raigترین نوع چندريختي به حساب ميآيد. معمولاً زمانی که گفته می‌شود شيئي چندريختي دارد، درواقع در 70% مواقع منظور همين پلي مورفيسم زيرنوع است. در اين نوع، اطلاعی از نوع يك شی تا زمان اجرای آن وجود ندارد. در اين نوع چندريختي، يك کلاس والد هست که يکسری کلاس‌های فرزند از آن ارث‌بری می‌کنند و متدهای اون کلاس که کدهای بدنه ديفالت دارند رو بصورت اختصاصی در کلاس‌های فرزند method overriding می‌کنند .C116

- parametric (overloading):

به‌طور خاص روشی را برای استفاده از يك تابع (کدهای يکسان) برای تعامل با چندین نوع داده مختلف فراهم می‌کند. اين نوع از پلي مورفيسم امكان استفاده از کدهای يکسانی را برای نوع داده‌های مختلف فراهم می‌کند.

مثلاً يك تابع با عنوان sum مينويسيم که اين تابع قابلیت جمع کردن هر نوع داده‌ای از جمله int, float, string ... رو دارد.

- ad hoc (compile-time):

التابع با نام يکسان برای انواع مختلف رفتار متفاوتی از خود نشان می‌دهند و به صورت موردي عمل می‌کنند.

مثلاً چندتا تابع با نام‌های يکسان مثل sum مينويسند که هر يك از اين توابع مخصوص يك نوع داده هست. درواقع ورودی تابع مشخص می‌کنه که اين تابع برای کدام نوع داده هست.

مثلاً اگر تابع sum رو کال کردیم و ورودی ایی از نوع int فرستادیم تابع sum مربوطه به int اجرا میشه و یا اگه ورودی رو string دادیم تابع sum مربوط به string اجرا خواهد شد و ... . این نوع چندريختی به دو دسته method overloading, operator overloading دسته بندی میشه. فلسفه چندريختی موردی با این صورت هست که گفته شد ولی پایتون اجازه تعریف توابع همنام رو بصورت دیفالت در یک سطح نمیده چون آخرین تابع همنام جایگزین تابع های قبلی میشه. مگر با ترفند هایی. این نوع چندريختی در Cpp مرسوم هست.

- coercion (casting):

تبديل مستقيم یک نوع داده به نوع ديگر است. اين نوع از پلي مورفيسم در برنامه نويسي زمانی اتفاق میافتد که یک نوع داده به نوع ديگر تبديل يا به اصطلاح Cast میشود. پيش از اين، چندريختی از طريق تعامل با نوع داده های مختلف به واسطه شی کلاس یا توابع انجام میشود. نوع یک شی زمانی قابل انتخاب بود که برنامه اجرا میشود. امكان اجرای یک تابع واحد روی انواع داده مختلف نیز وجود داشت.

برای ارائه مثالی ساده در این خصوص، میتوان تبدیل مقادیر عددی از نوع صحیح (int) به نوع اعشاری (double)، اعشاری شناور (float) و برعکس را نام برد. بسته به زبان برنامه نويسي، يا نوع را میتوان در زمان تعریف متغیر مشخص کرد و گاهی هم ممکن است برای نوع داده وجود دارد که مقدار متغیرها را به نوع ديگر تبدیل میکند. در ادامه مثالی برای درک بهتر پلي مورفيسم در برنامه نويسي به زبان های مختلف از نوع Coercion ارائه شده است. به طور کلی دو نوع Casting وجود دارد:

«تبديل نوع موقت ضمنی» (غیرصریح | غیر واضح) یا «Implicit Casting» با استفاده از خود کامپایلر انجام میشود.

«تبديل نوع موقت صریح» (آشکارا واضح) یا «Explicit Casting» با استفاده از const\_cast ، dynamic\_cast و سایر موارد انجام میشود.

## Duck typing

وقتی وارد اکوسیستم پایتون میشوید باید کدهای پایتونیک بنویسید. Python به مجموعه قواعدی گفته میشود که برنامه نویسان پایتون برای بهتر کدنوشتن پیشنهاد میدهند. در زیر در رابطه با سه موضوع مهم صحبت خواهد شد که میتوانید با آنها کدهای پایتونیک بنویسید.

مفهوم duck typing به زمانی اشاره میکند که نوع اطلاعات ورودی اهمیت ندارد و فقط نیاز است که اطلاعات ورودی ویژگی خاصی را داشته باشند. در زمانی که از duck typing در کد خود استفاده میکنید نیازی نیست که نوع آبجکت ورودی را بررسی کنید. فقط وجود متدها یا attribute خاصی را بررسی میکنید.

توی زبانی که از ساپورت میکند هر نوعی که شبیه به نوع اصلی باشد رو میتوانه اجرا کنه و اما در زبانی که ساپورت نمیکند باید دقیقاً از همون نوع بدیم تا متدها شو فراخونی و ... کنه.

مثالی که میتوان از duck typing در پایتون زد، فانکشن len() است. برای فانکشن len اهمیتی ندارد که چه نوع آبجکتی را میگیرد. تنها چیزی که اهمیت دارد، وجود متدها \_len\_ است. هر آبجکتی که متدها \_len\_ را در خود داشته باشد، میتواند به متدها len ارسال شود در غیر اینصورت پیغام خطای TypeError خواهد داد.

```
>>> len('amir')  
4  
>>> len([1, 'amir', 6])  
3
```

متدها len به طور کامل مفهوم duck typing را پیاده سازی کرده است. بدون اهمیت به نوع آبجکت، تعداد آیتم‌های درون آن را برمیگرداند.

اما مفهوم duck typing میتواند خطرناک باشد. اگر نتوانید به درستی تمام حالت‌های ممکن را کنترل کنید ممکن است در کدتان به مشکل بخورید. متدها len نمیتواند روی آبجکت‌های عددی کار کند:

```
len(373)
```

TypeError: object of type 'int' has no len()

در اینجاست که باید قبل از استفاده، بررسی کنید که آیا ویژگی که به دنبال آن هستید در آن آبجکت وجود دارد یا نه. اگر وجود داشت که میتوانید ادامه دهید اگر نه باید از ادامه عملیات جلوگیری کنید. LBYL = look before you leap

مفهوم LBYL توضیح میدهد که بهتر است قبل از انجام کاری بررسی کنید که آیا متدها وجود دارد یا خیر.

حالا در این کد بررسی میکنیم که آیا آبجکت ورودی دارای متدهای `len` هست یا نه: **C102**.  
در این کد تابع یک آبجکت بعنوان ورودی میگیرد که اگر آن آبجکت دارای متدهای `len` باشد عملیات `LBYL` ادامه میابد در غیر اینصورت پیغام مناسب چاپ میشود. درواقع ما در این کد از مفهوم استفاده میکنیم. یعنی قبل از اینکه روی `obj` کاری کنیم بررسی میکنیم که آیا ویژگی `len` را دارد یا نه.

اما باید بدانید که مفهوم `LBYL` آنچنان `pythonic` نیست و در اکوسیستم پایتون پیشنهاد میشود که از آن استفاده نشود و بجای آن از مفهوم بهتری با عنوان `EAFP` استفاده شود.

`EAFP = it's easier to ask for forgiveness than permission`

در خواست بخشش آسان‌تر از اجازه هست.

این مفهوم کاملاً برعکس `LBYL` هست. یعنی نیاز به بررسی نیست، اول کاری که میخوای رو انجام بده، اگر درست کار کرد که هیچی، اما اگر درست کار نکرد، معذرت خواهی کن.

در ادامه مثال قبل دیگر نیازی به بررسی وجود ویژگی نیست. ابتدا ویژگی رو فراخوانی میکنیم اگر وجود داشت که کدامان به درستی کار خواهد کرد، الس باید `exception` که رخ میدهد را کنترل کنیم.

مثال قبلی رو با استفاده از مفهوم `EAFP` باز نویسی میکنیم و به این صورت در میاد: **C103**.

در کد جدید دیگر وجود متدهای `len` رو بررسی نمیکنیم. مستقیماً کد را اجرا میکنیم، در صورت بروز خطای پیغام خطا مناسب را به کاربر نمایش میدیم.

در این مثال کد ما از دو نظر بهتر شده:

کد کوتاه‌تر شده پس خوانایی آن بیشتر شده.

سرعت کد بیشتر شده چون دیگر نیازی به دسترسی به آبجکت و بررسی اضافی نداریم.

در زبان‌های دیگری مثل `java` از `interface` برای این منظور استفاده میشود.

## Rules in oop

Important rules in object oriented programming:

a) everything in Python is object.

يعنى وقتی يك تابع تعريف ميکنيم، يك کلاس تعريف ميکنيم، در همهی اينها يك شى از کلاس مربوطه آن ايجاد ميکنيم .C104

در اوایل آموزش گفته شد که data type های مختلفی در پایتون داریم و علاوه بر آنها میتوانیم نوع داده‌های اختصاصی خودمون رو هم با استفاده از تعريف کلاس و نمونه سازی از آن ايجاد کنيم. همانطور که اشاره شد وقتی يك کلاس تعريف ميکنيم از کلاس type يك شئ ميسازيم و ميشه گفت يك کلاس تعريف شده يك شئ هست از کلاس type. درواقع میتوانیم بگیم کلاسی که ساختیم يك type هست چون شئ‌ایی از کلاس type هست. و در مفهوم معادلش يك لامبورگینی يك ماشین است.

پس وقتی میگیم يك کلاس ايجاد ميکنم يا يك type ايجاد ميکنم عملاً فرقی باهم ندارند و میتوانن بجای يکدیگر استفاده شوند. پس مفهوم کلاس برابر با مفهوم type هست.

همه کلاس‌ها در پایتون از کلاس object ارث بری میکنند

هر کلاسی يك شئ از کلاس type هست و همه کلاس‌ها از کلاس object ارث بری میکنند

b) each object must be an instance of at least one class.

c) a variable is a reference to an object.

a=3

b="shahin"

يعنى وقتی يك literal ايی به يك متغير انتساب ميدیم درواقع يك شئ از اون کلاس (built-in classes) ميسازيم(انتساب دادن هر literal ايی به متغير مثل يك برجسب، برای ارجاع به شئ هست نه چيز ديگه).

مثلاً در خط اول يك شئ‌ایی از کلاس integer و در خط دوم يك شئ‌ایی از کلاس string ساختیم. برخلاف کلاس‌هایی که خودمون ميسازيم و بصورت صريح هنگام نمونه سازی میگیم نمونه‌ای از فلان کلاس در اين متغير قرار بده (obj=A()) در کلاس‌های built-in به اين صورت و بطور غير مستقيم ازشون نمونه ميسازيم.

و اينکه مثلا وقتی میگیم  $a^*b$  درواقع از متدهای تعريف شده اون اشیاء بصورت غيرمستقيم استفاده ميکنيم. و علاوه بر استفاده غير مستقيم میتونم اصلاً بصورت مستقيم هم از متدهای مربوطه بصورت زير استفاده کنيم:

## concepts in oop

obj = ClassName

obj = ClassName()

تفاوتش این دو تا اینه که تو اولی اسم کلاس رو عوض میکنیم  
ولی تو دومی یک شئ از کلاس میسازیم

Adding attribute to the object manually outside the class:

objectName.attributeName = attributeName = car.color = "blue"

car.color: str = bule

میتوانیم از type hints هم استفاده کنیم:

مقدار هر attribute هر نوع داده‌ای میتوانه باشه از جمله int, string, ... یا یک تابع یا یک object از یه کلاس دیگه باشه.

: اولین آرگومانی که بصورت اتوماتیک به متده موردنظر ارسال می‌شود متغیر self variable هست. این متغیر به آبجکتی که از کلاس ساخته شده اشاره می‌کند. برای دسترسی به متدها و متغیرهای یک آبجکت باید از متغیر self استفاده کنید.

اگر بخوایم از طریق کلاس به متده مورد نظر دسترسی پیدا کنیم باید متغیر self را بصورت دستی ClassName.methodName(obj) = Car.move(obj) pass کنیم:

## property

دلایل استفاده از **property**: در زبان‌های دیگه برای حفظ امنیت، اعتبارسنجی، ارزیابی و ...، برخی خصوصیات، از طریق آنها رو ایجاد و با آنها تعامل پیدا میکنیم. که در اینصورت اون خصوصیات در حالت private قرار میگیرند.

**Note:** وقتی از **property** استفاده میکنیم، حین initializing `self.name` از طریق کارت ابجکت، property مربوطه رو کال میکنیم تا `set or get`, ... رو انجام بده. مثلًاً حین `_name` خصوصیت مورد نظر(`_name`) رو ست میکنه.

خصوصیت خصوصی هست و خود `name` پراپرتی ایی هست که با `_name` در تعامله.   
`season_10\packages\color.py`.

`class property(fget=None, fset=None, fdel=None, doc=None)`: returns a property attribute.

`fget` is a function for getting an attribute value. `fset` is a function for setting an attribute value. `fdel` is a function for deleting an attribute value. And `doc` creates a docstring for the attribute.

If given, `doc` will be the docstring of the property attribute. Otherwise the property will copy `fget`'s docstring (if it exists).

**Note:** هر ارجومانی که موقع ارجومان دهی به تابع `property` پاس میشه، تابع فقط همون قابلیت هارو برای attribute مورد نظر ست میکنه.

در این مثال که یک رفتار جدید به کلاس لیست اضافه کردیم و میتوانیم این رفتار رو به یک صفت با دکوراتور `property` تبدیل کنیم و مثل یک صفت بدون پرانتر فراخوانیش کنیم **C120**.

## Descriptor

فرض کنید در یک کلاسی اسم فادر، مادر و چایلد رو میگیریم(هر سه خصوصیت از یک جنس یعنی str هستند) و initializing میکنیم، ولی میخوایم با پروپرتی هر سه تا اسم رو ارزیابی، و فیلتر هایی برashون بزاریم که این ارزیابی و فیلترها در هر سه تا اتtribut یکسان هست. روش دیفالت این کار پیاده سازی متدهای ستر، گتر، و دیلیت برای هر اسم هست که اگر بخوایم برای هر خصوصیت(هر اسم) این سه تا متده را پیاده کنیم تعداد متدها زیاد میشه و باعث تکرار کد و ... میشه، که این بهینه نیست.

خب، با استفاده از توصیفگر میتوانیم کدهای property دیفالت این سه متده را override کنیم که میتوان برای هر سه خصوصیت استفاده کرده و دیگه نیازی نیست که برای هر اtribut، property اختصاصی رو پیاده کرد. چون با یک توصیفگر میتوانیم همه خصوصیتهایی که یکسان هستند رو ارزیابی کنیم.

دیسکریپتورها کلاس هایی هستند که حداقل یکی از متدهای `__get__`, `__set__`, `__delete__` رو customize میکنند تا property را override کنند. از دیسکریپتور فقط برای class attributes استفاده میشه.

فرضًا کلاس A رو یک کلاس دیسکریپتور و B رو یک کلاس نورمال در نظر میگیریم. و در بدنه کلاس B، بعنوان class attribute یک instance از کلاس A بنام father\_name و ... میک میکنیم و اگر با instance ای از کلاس بی، father\_name رو پرینت کنیم، خروجی خروجی متده `__get__` اورراید شده هست.

در حالی که اگر این متده override نشده بود، اگر پرینتش کنیم نتیجه همان خروجی ای رو که وقتی خود instance رو که از یک کلاس ساخته شده و پرینتش میکنیم که میگه در فلان جای حافظه هست رو میده.

در حقیقت کدهای property دیفالتی که برای همه اtribut های ستد شده رو با کلاس دیسکریپتور customize کرده و رفتارشون رو property decorator نیست .C131

خب تا اینجا همه چی به ظاهر اوکی هست و کدها بهینه شده ولی یک مشکلی که در بک هست اینه که اگه بگیم `print(p.__dict__)`، داندر dict کلاس Parent امپتی هست! دلیلش هم آینه که خصوصیات در `__dict__` کلاس دیسکریپتور سیو شده و اصولاً باید خصوصیت هر کلاس باید در `__dict__` همون کلاس سیو شده باشه. برای حل مشکل کدهارو تغیر میدیم و اینبار از طریق ارگومان

کلاس دیسکریپتور اسم هر class attribute را حین نمونه سازی بهش میدیم و اون سه تا متده استفاده از اون اسم، در \_\_dict\_\_ کلاس parent، ست، گت یا دیلیت رو انجام میده. پس از این به بعد خصوصیات در \_\_dict\_\_ خود کلاس parent سیو شده .C132

ولی روش بهتر از این استفاده از متده \_\_set\_name\_\_ در کلاس دیسکریپتور هست. متده \_\_set\_name\_\_ تفاوت خاصی با متده \_\_init\_\_ نداره و مثل اون یک خصوصیتی روی برای instance ست میکنه ولی فقط تنها تفاوتش آینه که دیگه نیاز نیست در کلاس parent اسم خصوصیت رو بعنوان آرگومان به کلاس دیسکریپتور در حین instantiating بدیم، زیرا این اسم رو متده \_\_set\_name\_\_ عز اتوماتیک برمیداره .C133

## Callable objects

Everything in python is an object, like function, class, etc. A function is an object of "function" class, or a class that we define it is an object of "type" class.

Now, some objects is callable like functions and classes. A callable object is any object that you can call using a pair of parentheses and, optionally, a series of arguments. Functions, classes, and methods are all common examples of callable object's is python. Besides these, you can also create custom classes by `__call__` method that produce callable instances.

`callable(obj)`: returns True or False for determine that the object is callable or no.

با استفاده از کال متدهای میتوانیم شی رو دوباره فراخوانی کنیم و خصوصیتی رو تغیر بدیم که البته توانایی‌های بیشتر از این رو داره **C119**.  
Callable object در ایجاد کلاس دکوراتور هم کاربرد دارد.

## Type of methods and attributes

instance method: رایج ترین متدهایی هستند که self رو به عنوان ارگومان میگیرند و با تعامل دارند - همچنین این متدها میتواند با کمک ویژگی self.\_class به خود کلاس (methods, class attributes) نیز دسترسی داشته باشند.

:class method و cls رو که کلاس هست بصورت اوتوماتیک میگیرند و با خود کلاس instance attributes (class attributes, methods) در تعامل هستند. و به methods دسترسی ندارند از دکوراتور @classmethod استفاده میکنند - معمولاً با اسم کلاس کال میشن ولی با instance هم میشه کالش کرد.

با cls به نام کلاس (\_name\_) و class attribute ها دسترسی داریم. عموما از کلاس متدها برای ایجاد factory method استفاده میشود. همانند یک نمونه از کلاس برای use case های مختلف رو برمیگردانند. constructors

:static method ورودی خاصی مثل self or cls ندارند و با دکوراتور @staticmethod فعال میشن. این متدها به هیچ یک از instance attribute, class attribute نه نیاز ندارند و همچنین دسترسی هم به آنها ندارند و برای استفاده از آنها نیاز به نمونه سازی از کلاس نیست زیرا میتوان با نام کلاس کالش کرد ولی با instance هم میشه کالش کرد.

متدهای خاصی کلاس خود را میشناسد و از طریق آن شی کلاس خود را نیز میشناسد.  
متدهای کلاس خود را میشناسد.  
متدهای ایستا، کلاس یا نمونه خود را نمیشناسد.

:magic method این متدها را هرگز بصورت مستقیم کال نمیکنیم زیرا این متدها بصورت غیرمستقیم توسط پایتون در پشت صحنه اجرای کدها کال میشه. این در کلاسهای built-in پیاده‌سازی شده‌اند. اما میتوانیم هر زمان که نیاز داشتیم این متدهارو override کنیم.

[seasons\\_10/packages/comment.py](#)

## Type of attribute:

- class attribute
- instance attribute

The reason for replacing attribute classes with instance attributes:

`self.num_base_calls += 1` is basically `self.num_base_calls = self.num_base_calls + 1`. It first falls back to reading from the class attribute, but ends up setting `self.num_base_calls`. If you want to increment a class attribute, you need to explicitly increment it on the class object(`BaseClass.num_base_calls+=1`), not `self` C112.

## object initialization

در زبان‌هایی که از شئ گرایی پشتیبانی می‌کنند یک متدهای بنام constructors وجود دارد که مقداردهی اولیه آبجکت‌ها را انجام میدهد. معادل این متدهای در پایتون متدهای `__init__` هست. به محض ساخت یک آبجکت از کلاس، متدهای `new` کال می‌شود و یک شی از کلاس می‌سازد و سپس توسط متدهای `__init__` مقداردهی‌های اولیه انجام می‌شود و سپس متدهای `new` اون شی را `return` می‌کنند و در متغیر قرار می‌گیرند.

متدهای سازنده در زمان ارث بری: اگر در کلاس فرزند متدهای سازنده نداشته باشیم، متدهای سازنده کلاس والد فراخوانی می‌شود.

**Class's dunder methods:** they are defined by built-in classes in Python and commonly used for operator overloading

`__class__`:

`__delattr__`:

`__dict__`:

`__dir__`:

`__doc__`:

`__eq__`:

`__format__`:

`__ge__`:

`__getattribute__`:

`__getstate__`:

`__gt__`:

`__hash__`:

`__init__`: used to initialize objects of a class. It is also called a constructor.

`__init_subclass__`:

`__le__`:

`__lt__`:

`__module__`:

`__ne__`:

`__new__`: whenever a class is instantiated `__new__` and `__init__` methods are called.

`__new__` method will be called when an object is created and `__init__` method will be called to initialize the object. The `__new__` method is defined as a static method which requires to pass a parameter `cls`. `cls` represents the class that is needed to be instantiated, and the compiler automatically provides this parameter at the time of instantiation.

`__reduce__`:

`__reduce_ex__`:

`__setattr__`:

`__sizeof__`:

`__repr__`: to get called by built-in `repr()` method to return a machine readable representation of a type. provides an official representation of string.

معمولاً اگر خروجی `repr` رو به تابع `eval` بدهیم میشه اون ابجکت رو بازسازی کرد.

`__str__`: to get called by built-in `str()` method to return a readable representation of a type for end user. provides an unofficial representation of string.

by default, in IDE, print function behavior is `__str__`'s method behavior. but in terminal it is reverse.

وقتی در یک کلاس متدهای `__str__` و `__repr__` را تعریف نکرده باشم تابع `print` بصورت دیفالت از متدهای استفاده میکنه.

مثال ملموس برای متدهای `repr` و `str`: C110

`__subclasshook__`:

`__weakref__`:

`__del__`: deletes object's.

`__delete__`: deletes attribute from father class.

`__getattr__`: returns an attribute value.

`__hasattr__`: checks if a certain attribute exists.

`__setattr__`: ویژگی موجود را تغییر می‌دهد یا ویژگی جدیدی ایجاد می‌کند:

`__dict__`: returns a dictionary of attributes and their values.

`__dir__`:

## Limit attributes

عز عه دیفالت همهی خصوصیات یک شی در یک دیگشتری (`obj.__dict__`) در دسترس هست و با استفاده از متدهای `__init__` اینیت حین instantiating خصوصیات مورد نظر را به شئ مورد نظر اضافه میکنیم ولی راههای دیگه ای هم برای اینکار وجود داره. مثلًا بیرون از کلاس میگیم `obj.x=3` یا اینکه به دیگشتری داندر دیکت خصوصیتی رو میتوانیم اد کنیم.

ناو، میخوایم که اضافه کردن هر خصوصیتی به شی مورد نظر با محدودیتهایی انجام بشه، که این کار رو با `__slots__` انجام میدیم.

پسری اسامی ایی که میخوایم اسامی خصوصیات شئ باشه رو در بدنه کلاس داخل یک تاپل به داندر اسلات ست میکنیم و از این به بعد فقط خصوصیتی که اسمش از قبل داخل داندر اسلات هست رو میتوانیم به شی مورد نظر اد کنیم.

و همچنین داندر اسلات دیگشتری رو نیز غیرفعال میکنه [C128](#).

و یک مورد دیگه اینکه در دو تا کلاسی که ساب کلاس از تاپ کلاس ارث بری کرده. اگر در تاپ کلاس داندر اسلات رو ست کنیم در حالی که در بدنه ساب کلاس داندر اسلاتی دیفاین نشده، `m.__dict__` رو با اینستنسی از ساب کلاس فراخونی کنیم داندر دیکت ساب کلاس غیرفعال نشده ولی خالی میباشد.

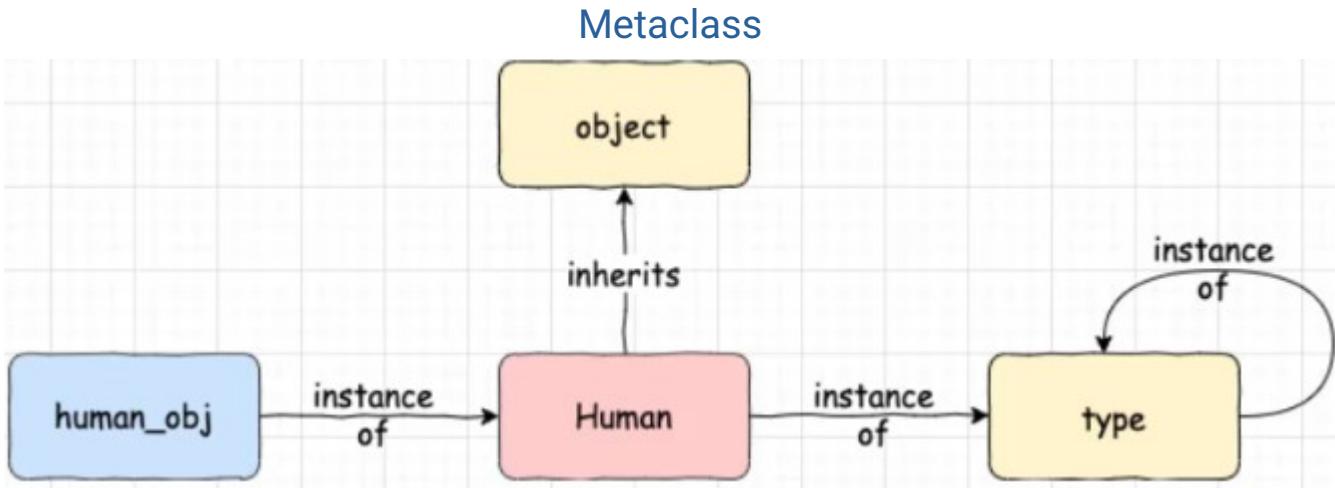
و اگر بخوایم یک خصوصیتی به شئ ساب کلاس اضافه(`m.c=3`) کنیم اگر اسمش در بین عناصر داندر اسلات تعریف شده در بدنه تاپ کلاس نبود به داندر دیکت ساب کلاس اضافه میشه ولی اگر بود به که به داندر دیکت تاپ کلاس اضافه میشه که اونم دیسیبل میباشد [C129](#).

و یک حالت دیگه اینکه اگر برای ساب کلاس هم داندر اسلات ست کنیم و بعد بخوایم به شئ ساب کلاس خصوصیتی اضافه کنیم(`m.e=4`) در این حالت باید اسم اون خصوصیت یا در داندر اسلات تاپ کلاس یا در داندر اسلات ساب کلاس وجود داشته باشه یا در داندر اسلات تاپ کلاس، وگرنه خصوصیت جدید ست نمیشه [C130](#).

`__slots__ advantage:`

thrift in memory

Quick access to attributes



همهی کلاس‌ها، چه کلاس‌هایی built-in، چه کلاس‌هایی که ما تعریف میکنیم بصورت اتوماتیک از کلاس object ارث بری میکنند و علاوه بر این، نمونه‌ای از کلاس type نیز هستند.

For example, we have a class named M, and if the objects that are instantiated from class M are a class, so M is a metaclass.

A metaclass is a class that allows for other classes to be instantiated as objects of the metaclass.

بنابراین به کلاس‌هایی که نمونه‌های ساخته شده از آن کلاس باشند متاکلاس گفته میشه.  
همه نوع داده‌های پایتون یک نمونه از کلاس type هستند.

### Creating and Using a Metaclass

Here we create a metaclass called ExampleMetaClass. Then we create a class called SubClass that takes ExampleMetaClass as a keyword argument. This is how SubClass is instantiated as an object of ExampleMetaClass. We can see in the output of the code that the class from which SubClass is created is  
`<class '__main__.ExampleMetaClass'>`.

We also see in the output that the type of our ExampleMetaClass is `<class 'type'>`. So ExampleMetaClass is an instance of the type class here, and is therefore itself instantiated from the metaclass type.

Whenever a class is created, even if the metaclass keyword isn't defined, it is created from a metaclass. The default metaclass is the type metaclass-C123.

## Why Use Metaclasses?

Metaclasses allow for code not only to manipulate data, but to manipulate classes. Often this change happens when an object of the class is instantiated. Using metaclasses also helps to abstract our code, making it more readable and helping to reduce the amount of code written by avoiding repetition in code.

## Dataclass

همان‌طور که از نام آن پیداست هدف دیتا کلاس ارائه یک ساختار ساده و راحت برای ذخیره و بازیابی اطلاعات هست. با دیتا کلاس برخی از کارهای ساخت کلاس که بصورت تکراری هست رو به دیتا کلاس می‌سپاریم. در دیتا کلاس متدهای `eq`, `init`, `repr` بصورت پیشفرض پیاده شده اند. برای پیاده‌سازی دیتا کلاس از دکوراتور `dataclass` استفاده می‌کنم. در بدنه دیتا کلاس فیلدها (معادل `class attribute`) بصورت اجباری با `type hints` تعریف می‌شن. همان‌طور که ذکر شد متدهای `post_init` بصورت دیفالت پیاده شده ولی یک متدهای `__post_init__` وجود دارد که می‌توانیم پیادش کنیم تا بعد اینکه متدهای `init` دیفالت ران شد متدهای `post_init` ران بشه و یسری مقادیر دیگری رو هم سرت کنه [C134](#).

ارسال مقادیر به متدهای `post_init`: برای اینکار باید اسم اون خصوصیتی که می‌خوایم ارسال کنیم رو در زیر فیلدها با `type hint` و بعنوان پارامتر در متدهای `post_init` بنویسیم و سپس بقیه عملیات بسته به خواسته در بدنش پیاد می‌شه. و برای این مقدار رو حین `instantiating` به همراه آرگومان فیلدها، آرگومان خصوصیت مورد نظر که می‌خوایم ارسالش کنیم رو هم پاس می‌کنیم. بعد اینکه `gender` در اینجا یک خصوصیت نیست و صرفاً برای ارسال مقدار به متدهای `post_init` پیاده شده. و دکوراتور دیتا کلاس یسری مقادیری دیفالت داره که در صورت نیاز می‌توانیم تغییر بدیم. مثلًاً اگر اگر فالس باشه می‌توانیم مقدار یک فیلدر و چنج کنیم ولی اگر `true` باشه نه. چون با این کار به [C135](#) تبدیلش کردیم `immutable`

Make class attribute: [C136](#).

Inheritance in data classes: [C137](#).

## 11) Error management

target is the management a set of errors that may be happens in run program  
type of errors:

- syntax errors
- exceptions: errors that happens in runtime.

try:

    statements ...

exception [(exception name1, exception name2, ...)]:

    actions that do after exception.

exception:

    print('unknown error!')

else:

    print('else!')

finally:

    print('finally')

بدنه else زمانی اجرا میشه که هیچ استثنایی رخ نداده باشه و بدنه finally نیز در هر صورت اجرا میشه.

## Built-in functions

access to list of builtins names: C80.

Function	Description
<code>sorted(obj, reverse=True)</code>	sorts anything.
<code>zip(obj1, obj2)</code>	zips two object C10.
<code>min(obj, default=1)</code>	returns minimum obj- if obj was null returns default value.
<code>max(obj, default=5)</code>	returns maximum obj- if obj was null returns default value.
<code>sum(obj, start=10)</code>	returns sum of obj + start value.
<code>dir(obj)</code>	Returns all methods and attributes an object.
<code>range(start, stop, step)</code>	
<code>eval()</code>	Used to parse an expression as an argument(as "string") and then execute it.
<code>isinstance(a, b)</code>	a is instance from b.
<code>setattr(obj, attr, vle)</code>	Sets the value of the specified attribute of the specified object.
<code>object</code>	Required. An object
<code>attribute</code>	Required. The name of the attribute you want to set.
<code>value</code>	Required. The value you want to give the specified attribute

<https://backendbaz.ir/python-built-in-function-references/>

## Modules

### random

- random

باذه اعداد تصادفی بصورت دیفالت بین ۰,۱ هست ولی میتوانیم این بازه رو دست کاری کنیم. C17.

- methods

- uniform(5, 10)

alternative method for change range.

- randint(5, 10)

returns specified range as integer number data type.

- randrange(start, stop, step)

- choice()

choice random of a object, for example chooses an element of a list.

- sample(obj, 3)

choosing from an object and convert it to a list.

- shuffle(obj)

shuffling a object

- seed(number)

در صورت استفاده از این تابع عدهای random تولید شده براساس این عدد داده شده خواهد بود که همیشه قابل دسترسی هست.

در صورتی که از seed استفاده نکنیم بصورت دیفالت زمان سیستم جایگزین آن خواهد بود. C18.

## OS

getcwd	Returns the current working directory
repr(os.linesep)	فهمیدن اینکه کدام کارکتر ها برای جدا کننده بین خطوط استفاده میشند (در هر OS دیفرانس هست).
path.exists(x)	با مسیری که بهش داده میشی چک میکنی که آیا همچین فایلی هست یا نه.
remove(x)	Removes the file via its path or name

## importlib

فرضًا یه مقادیری رو مثلًا در ماژول تغیر دادیم و بعدش خواستیم اون مقادیر به حالت دیفالت خود برگرداند میتوانیم از ماژول importlib بصورت زیر استفاده کنیم تا ماژول reload بشه.

```
test2.py
1 def pow(n):
2     return n ** 2
3
4
5 name = 'reza'
6 print('hello')
7
```

```
test1.py
1 import test2
2 import importlib
3
4 print(test2.name)
5 test2.name = 'ali'
6 print(test2.name)
7
8 print(20 * '-')
9 importlib.reload(test2)
10 print(test2.name)

Run
C:\Users\araz> hello
reza
ali
-----
hello
reza
Process finished.
```

## re

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

<u>findall</u>	Returns a list containing all matches.
<u>match</u>	Returns a Match Object if there is a match only at the beginning of the string
<u>search A</u>	Returns a Match Object if there is a match anywhere in the string
.span()	Returns a tuple containing the start-, and end positions of the first match occurrence.
.string	Returns the string passed into the function.
.group()	Returns the part of the string where there was a match.
.start()	Returns the first position of the matching string
.end()	
<u>split</u>	Returns a list where the string has been split at each match.
<u>sub</u>	Replaces one or many matches with a string.

## A

**Match object:** a match object is an object containing information about the search and the result.

**Note:** if there is no match, the value **None** will be returned, instead of the Match Object.

## Metacharacters:

Chr	Description	Example	Example description
[]	A set of characters	[a-m]	Find all lower case characters alphabetically between "a" and "m"
\	Signals a special sequence (can also be used to escape special characters)	\d	Find all digit characters
.	Any character (except new line character)	he..o	Search for a sequence that starts with "he", followed by two(any) characters, and an "o"
^	Starts with	"^hello"	Check if the string starts with 'hello'
\$	Ends with	planet\$	Check if the string ends with 'planet'
*	Zero or more occurrences	he.*o	Search for a sequence that starts with "he", followed by 0 or more (any) characters, and an "o"
+	One or more occurrences	he.+o	Search for a sequence that starts with "he", followed by 1 or more (any) characters, and an "o"
?	Zero or one occurrences	he.?o	Search for a sequence that starts with "he", followed by 0 or 1 (any) character, and an "o"
{}	Exectly the specified number of occurrences	he.{2}o	Search for a sequence that starts with "he", followed exctly 2 (any) characters, and an "o"

	Either or	"falls  stays"	Check if the string contains either "falls" or "stays"
()	Capture and group		

...

## time

gmtime	Convert seconds to hour,minute,second.

## Logging

«لاگ کردن»، ثبت وقایع یا همان Logging در پایتون کمک میکند تا درک بهتری از برنامه نویسی داشته باشید و به وسیله آن سناریوهایی ارائه می‌شوند که ممکن است دولوپ در حین دولوپ تا به حال به آن‌ها فکر نکرده باشد.

Log‌ها به دولوپ‌ها این امکان را میدهند که بطور دائم جریانی که در یک برنامه در حال وقوع است را با دقت مضاعف دنبال کنند.

other

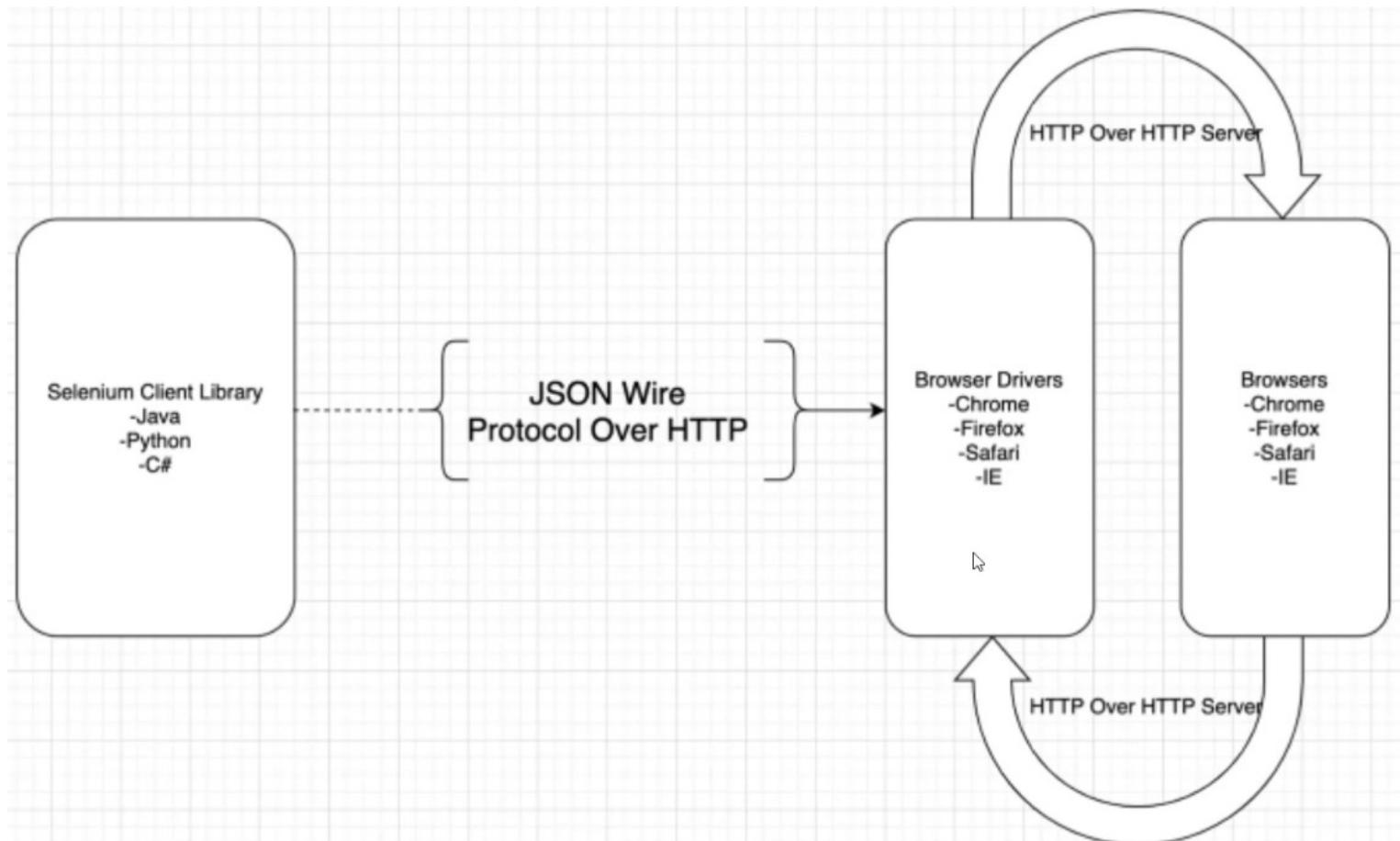
from pprint import pprint

برای نمایش هر عنصر لیست زیر یکدیگر

: web automation به شاخه‌ای گستره‌ای از فعالیت بصورت بات در سایت یا وب اپلیکیشن‌های مختلف گفته می‌شود (کپی کردن عملیات انسان با سایتها). به کمک web automation دو رویداد اصلی استخراج اطلاعات و اکشن‌های مختلف را می‌توانیم انجام بدیم.

یکی از زیر شاخه‌های web automation web scraping است؛ یعنی یک رباتی با یک هدف خاصی بسازیم که وارد سایتی بشه و اطلاعات خاصی را استخراج کنه.

: Chrome driver



## methods

driver=webdriver.Chrome()

get(url)	Opens the url
quit()	Quits the browser