



# آشنایی با کامپیوتر

جلسه اول

بسم الله الرحمن الرحيم



کارگاه مبانی برنامه‌نویسی - دانشکده مهندسی کامپیوتر دانشگاه امیرکبیر



اولین چیزهایی که نوزادان یاد می‌گیرند این است که چطور باید پدر و مادر خود را صدا بزنند و چطور می‌توانند با کلمات با دنیای اطراف خود صحبت کنند.


در دنیای علم هم چنین قانونی حکم می‌کند و هر کسی که می‌خواهد دانشی را کسب کند باید در ابتدا به کلمات دنیای آن تا حدی به تسلط رسیده باشد تا بتواند با دیگر افراد آن دنیا ارتباط برقرار کند.

دانش کامپیوتر هم مثل بسیاری از علوم دیگر است که برای قدم گذاشتن در آن نیاز داریم تا با اصطلاحات اولیه‌ی آن آشنا شویم. به علاوه باید بدانیم که کامپیوتر چطور کار می‌کند تا بتوانیم از توانایی‌های آن در جهت اهدافی که داریم بهره‌برداری کنیم. پس در این دستورکار هم می‌خواهیم با تعدادی از اصطلاحات آشنا شویم و هم بفهمیم که کامپیوترها چطور کار می‌کنند.


# فهرست



## مقدمه

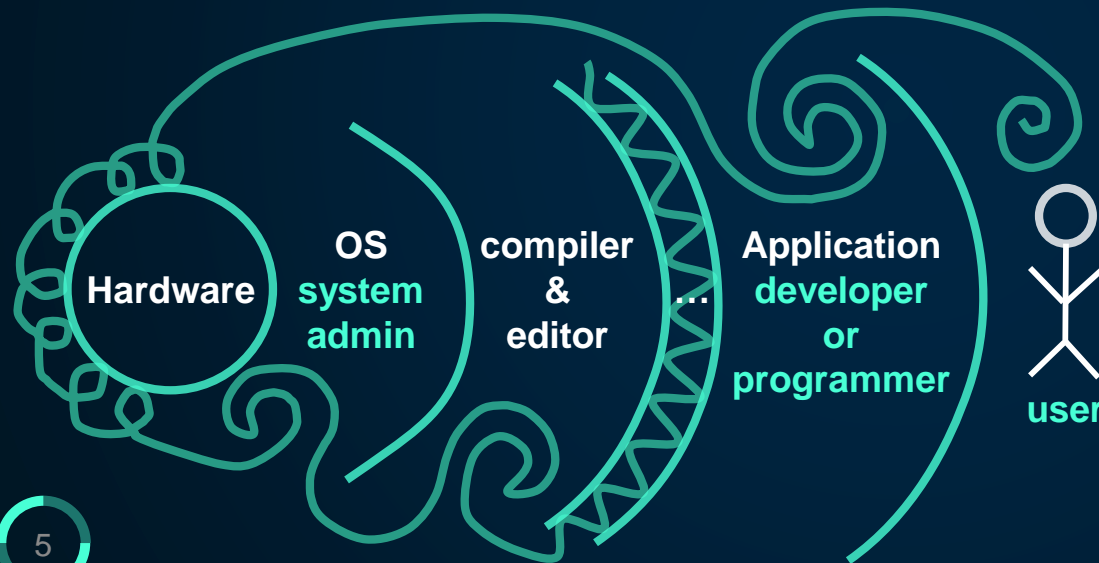
چرا ما از کامپیوترها استفاده می‌کنیم؟ چه چیزی باعث شده تا کامپیوترها امروزه تبدیل به عضو جدایی ناپذیری از زندگی همه‌ی شده باشند؟ 

برگ برنده‌ی کامپیوترها، سرعت غیر قابل مقایسه با سرعت انسان در انجام عملیات‌های مورد نیاز اوست و این امر به انسان کمک می‌کند تا با سرعت بیش‌تری به گسترش حوزه‌های علمی خود بپردازد.

البته ذهن ما، شاید توانایی انجام بلیون‌ها عملیات و پردازش را در سرعتی به سرعت کامپیوتر نداشته باشد، اما آنچه واضح است این است که این وسیله‌ی شگفت‌انگیز در نهایت حاصل خلاقیت و ذهن بشر و ساخته دست اوست، پس آنچه برای ما قابل فهم و درک است چگونگی ساخت و پیشرفت کامپیوترهاست. 

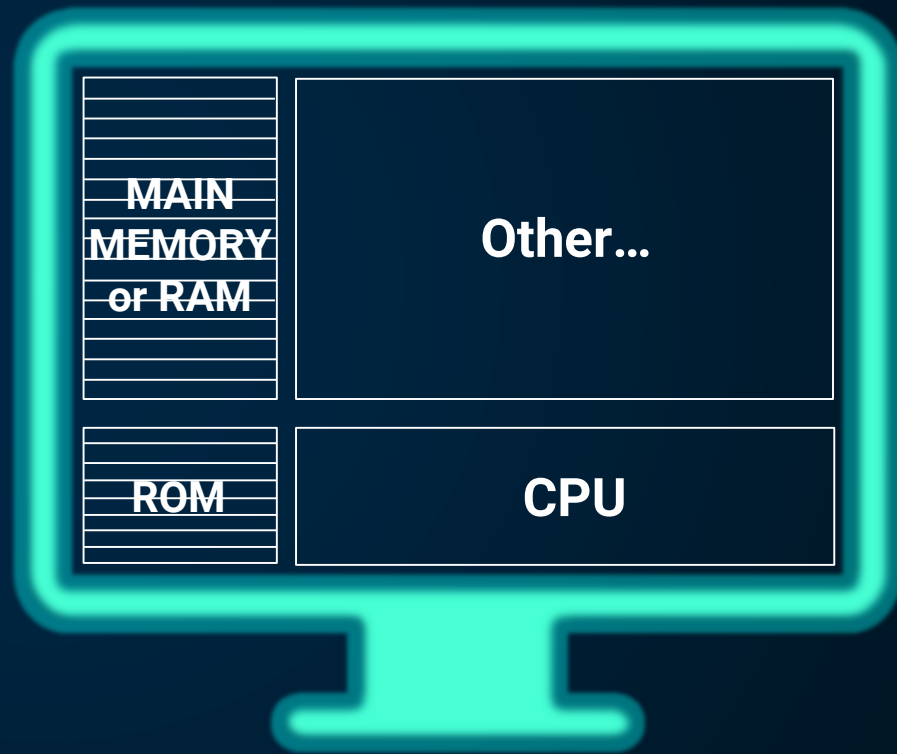
ما برای رسیدن به چنین درک و دانشی، از طریق ایجاد دید کلی، می‌توانیم روی طریقه‌ی کارکرد کامپیوترها تسلط یابیم.

این تسلط مرحله به مرحله اتفاق می‌افتد تا شما یک مهندس کامپیوتر شوید. روندی که شما در دانشگاه طی می‌کنید مشابه تصویر زیر است.

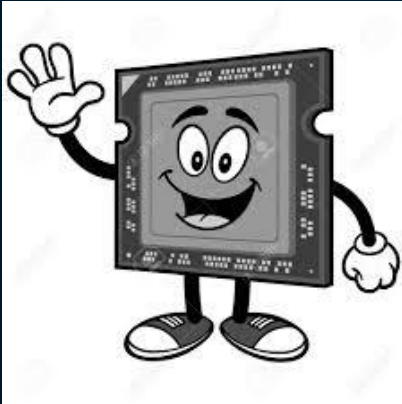
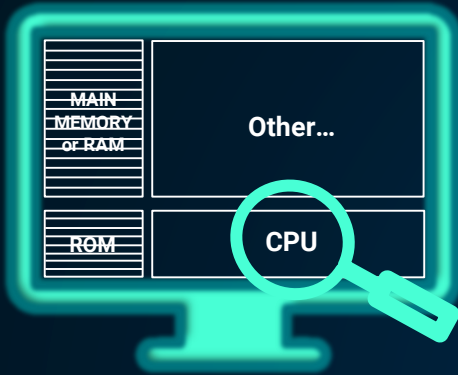


در طی این دستورکار یک دید کلی به بخش‌های مختلف این شکل پیدا خواهید کرد و با اصطلاحات پایه‌ای هر کدام آشنا خواهید شد. برای شروع بیاید بیشتر با ساختار کامپیوترها آشنا بشویم...

# رادیولوژی



## به عنوان اولین بخش از CPU شروع می کنیم.



### CU (Control Unit)

کنترل کل CPU دست منه. من میگم کی بیا کی بره. کی بشینه کی بلند شه. کی می تونه وارد CPU بشه و کی می تونه ازش خارج بشه. هیج کی اینجا بدون اجازه ی من آب نمی خوره.

### ALU (Arithmetic and Logic Unit)

من وظیفه ی انجام محاسبات و پردازش هایی رو دارم که CU به من دستور داده باشه. هر وقت شما جمع، تفریق، ضرب و خیلی از عملیات های دیگه ای رو درخواست کنین من حاصل اون ها رو براتون حساب می کنم و میدم به CPU تا به دست شما برسونه.

### Special Floating Point processors

من اینجا هستم تا به دوستم ALU کمک کنم. اون اکثرا محاسبات اعداد اعشاری رو نمی تونه انجام بده. برای همین من اومدم که حواسم به این اعداد باشه.

### Registers

ما وظیفه ی ذخیره ی اطلاعات رو داریم.

چون CPU حافظه نداره و اطلاعاتش رو از RAM می گیره.

اما اگر همه چی رو بخواد بگیره خیلی کند می شه.

پس ما اینجا ییم برای ذخیره ی یه سری اطلاعات که...

...

نیازه خیلی سریع در دسترس باشه و دیگه CPU معطل نشه.

✓ CPU را می‌توان مانند مغز یک کامپیوتر یا فرماندهی اصلی آن دید. زیرا همانطور که متوجه شدید، CPU جایی است که همه‌ی پردازش‌های کامپیوتر در آن انجام می‌شود و تمام دستوراتی که اجرا می‌شوند توسط CPU داده شده‌اند.

✓ CPU ها انواع مختلفی دارند که هر کدام از نظر سرعت، قیمت، کارایی و ... با هم تفاوت دارند.



آیا می‌توانید نوع CPUی دستگاه خود را بگویید؟



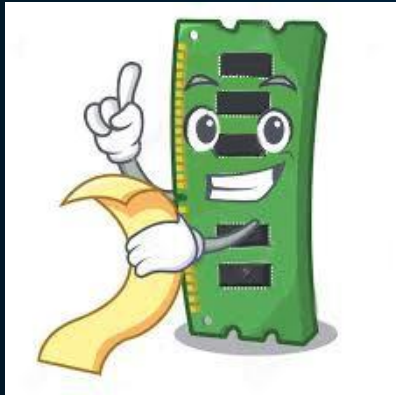
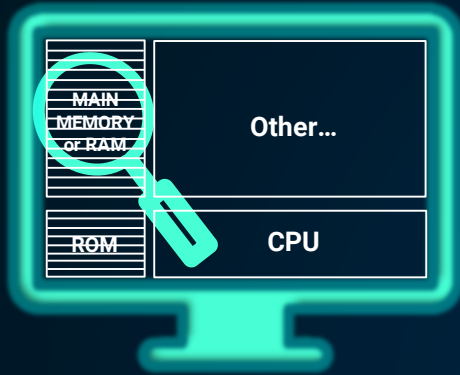


✓ هر CPU برای اینکه در انجام وظایف خود موفق عمل کند از چند بخش تشکیل شده است که مدیریت این بخش‌ها در نهایت توسط CU یا واحد کنترل انجام می‌شود.

✓ یکی از این بخش‌ها ALU<sup>۱</sup> یا واحد پردازش است که وظیفه‌ی انجام محاسبات را دارد. پس هرگاه نیاز به عملیات‌هایی مثل جمع، تفریق، ضرب و سایر عملیات‌های محاسباتی و منطقی داشتیم، ALU به عنوان یکی از سرداران CPU دستور اجرای آن عملیات و ورودی‌های مورد نظر را دریافت می‌کند و خروجی را دوباره به CPU تحویل می‌دهد.

✓ در CPU چندین ثبات<sup>۲</sup> نقش ایفا می‌کنند تا اطلاعات مورد نیاز CPU را به طور موقت ذخیره کنند. به عنوان مثال اگر قرار باشد دو عدد که در حافظه قرار دارند با هم جمع شوند، ثبات‌ها مقدار آن‌ها را ذخیره می‌کنند تا این مقادیر را به ALU بدهند و خروجی محاسبه شود. اگر ثبات‌ها نبودند، هر بخش از CPU مثل ALU یا سایر بخش‌هایی که به اطلاعات حافظه نیاز دارند، مجبور بودند که به طور مستقیم از حافظه، اطلاعات را دریافت کنند و در این حالت پیچیدگی سخت‌افزار ما خیلی بالا می‌رفت، چون به جای یک مسیر از CPU به حافظه، چندین مسیر وجود داشت.

## بخش بعدی Main Memory یا RAM است.



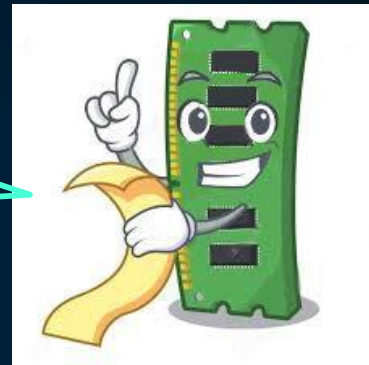
0	
1	
2	
3	
4	
...	
...	
...	
	...
...	
$2^n - 2$	
$2^n - 1$	

من از تعداد زیادی سلول‌های کوچولو (memory cell) برای ذخیره‌ی اطلاعات و دستورات تشکیل شدم.

همونطور که میدونی دنیای ما از ۰ و ۱ تشکیل میشه و همه‌چی تو مبنای ۲ه. برای همین که من  $2^n$  تا سلول حافظه دارم.

هر برنامه‌ای که قرار باشه اجرا بشه، اول من باید یه دور بخونمش و توی خودم ذخیره کنم تا بتونم دستورات و اطلاعات لازم رو به CPU بدم.

یک چیز مهم دیگه در مورد من اینه که اطلاعات تا زمانی یاد من می‌مونه که من رو به برق وصل کرده باشی. اگر برق رو قطع کنی یا کامپیوتر رو خاموش کنی، همه چی رو فراموش می‌کنم.



RAM یا **Random Access Memory** گونه‌ای از حافظه برای ذخیره‌سازی داده‌هاست که اجازه می‌دهد

فایل‌ها در مدت زمانی کوتاه نوشته و خوانده شوند. RAM به سیستم شما اجازه می‌دهد با سرعت بالا به داده‌های مورد نیاز دسترسی داشته باشد و در نتیجه تأثیر بالایی بر سرعت عملکرد سیستم شما دارد. یک نکته‌ی مهم RAM (همانطور که خودش هم گفت!) این است که یک حافظه موقتی است و پس از هر بار راه‌اندازی دوباره سیستم عامل تمامی داده‌های ذخیره شده روی آن پاک می‌شود و به همین دلیل از این نظر در مقابل **حافظه‌های غیر فرار**<sup>۱</sup>، قرار دارد.

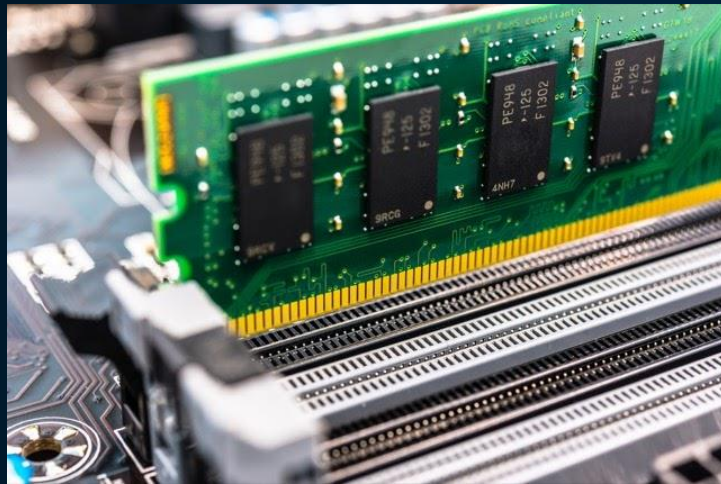


نکته‌ی دیگر در مورد آن‌ها، **Random**

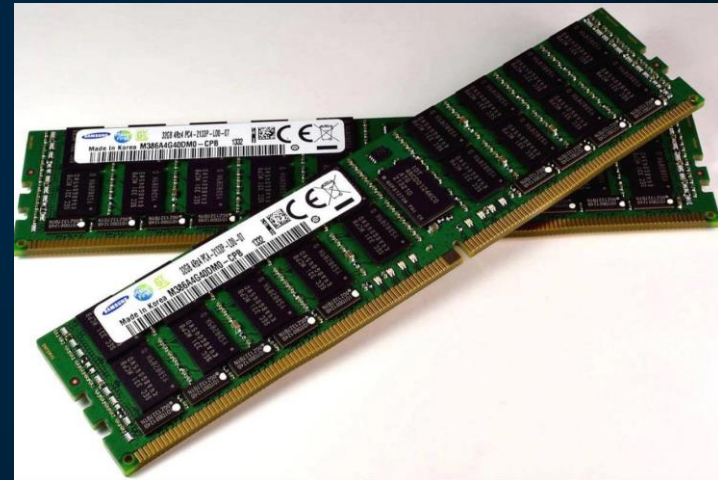
**Access** بودن آن‌هاست. دقت کنید که

این به معنای رندوم و تصادفی بودن

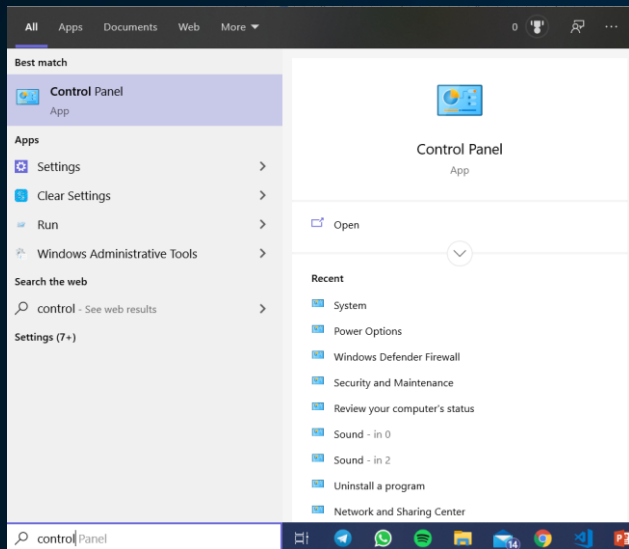
محتوای آن‌ها نیست!!



رندوم، صفتی برای نحوه‌ی **دسترسی** به این حافظه‌هاست و معنای آن این است که برای ذخیره‌ی اطلاعات در این حافظه‌ها، لزوماً با یک ترتیب همیشگی این اتفاق رخ نمی‌دهد.



## آیا می دانید چه میزان رم روی سیستم شما نصب شده؟



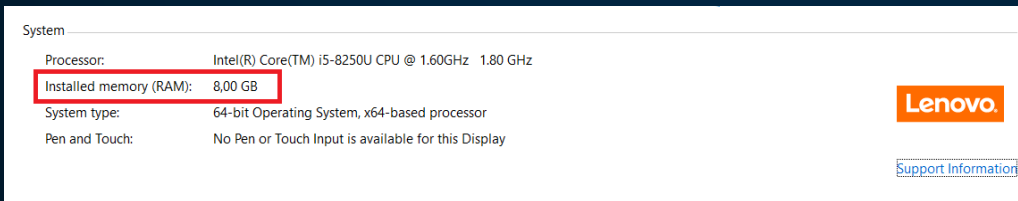
برای فهمیدن این موضوع در بخش Search لپ تاپ خود عبارت control panel را تایپ کنید.

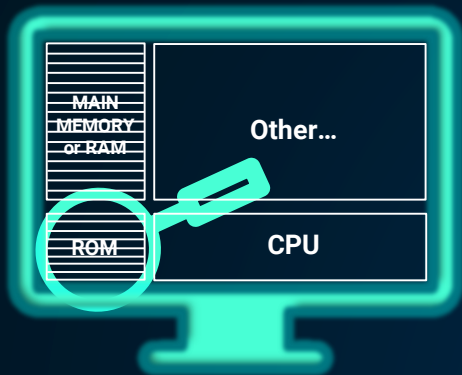


بعد از باز شدن پنجره‌ی مربوط به آن، ابتدا روی بخش System and Security و سپس روی System کلیک کنید.



در بخش زیر می‌توانید میزان حافظه‌ی RAM خود را مشاهده کنید.





0	
1	
2	
...	
	...
$2^a-2$	
$2^a-1$	

## بخش بعدی ROM است.

منم مثل RAM یک نوع حافظه هستم، اما تفاوتم با RAM در اینه که با قطع شدن برق اطلاعات از روی من پاک نمی‌شه.

حافظه‌های **ROM** به راحتی قابل اصلاح نیستند، بنابراین آن‌ها برای ذخیره‌ی داده‌هایی که برای مدت طولانی نیاز به اصلاح ندارند، مناسب هستند. حجم این حافظه‌ها زیاد نیست و تنها برای ذخیره‌ی داده‌های محدودی به کار می‌روند. هنگام روشن شدن کامپیوتر، پردازنده یا همان CPU برای **راه اندازی سیستم** از ROM استفاده می‌کند.

به نظر شما چه اطلاعاتی ممکن نیاز به تغییر نداشته باشند و باعث لزوم وجود این بخش از کامپیوتر شده‌اند؟



یک بار دیگر وارد بخش search شوید و عبارت cmd را چاپ کنید.

داخل پنجره‌ای که باز شده، یک صفحه‌ی مشکی می‌بینید با نوشته‌هایی مشابه تصویر زیر:

```
Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\bhrka>_
```



چیزی که می‌بینید پشت صحنه‌ی **سیستم‌عامل** شماست. هر دستوری که شما به کامپیوتر می‌دهید (مثلا با کلیک کردن) و هر کاری که برنامه‌ها انجام می‌دهند را می‌توانید در این پنجره هم با تایپ کردن یک سری دستورات (commandها) انجام دهید.



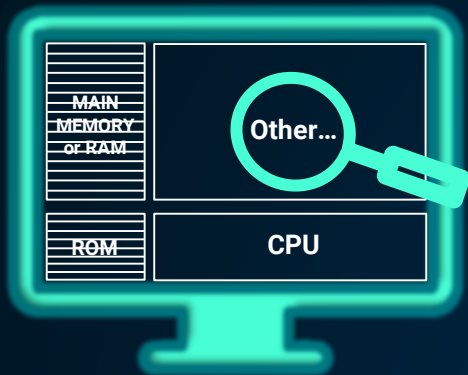


**سیستم عامل** در حقیقت یک برنامه است که هر بار با روشن شدن کامپیوتر شما شروع به کار می کند. اگر قرار بود این برنامه هر بار داخل RAM ذخیره می شد، شما بعد از هر بار روشن کردن کامپیوتر باید دوباره از اول **سیستم عامل** خود را (برای مثال همین سیستم عامل ویندوز) نصب می کردید، بعد برنامه هایی که لازم داشتید را روی آن نصب می کردید، کارهای خود را انجام می دادید و با همه ی این برنامه ها خدا حافظی می کردید!!!

چون همانطور که گفته شد، اطلاعات داخل RAM با خاموش شدن سیستم از بین می رود.

پس الان متوجه شدین که چرا وقتی کامپیوتر  
داره روشن می شه، CPU برای راه اندازی سیستم  
از ROM استفاده می کنه (:





## Secondary Storage

تا الان با دو نوع حافظه آشنا شدیم و وظیفه‌ی هر یک را متوجه شدیم. به نظر شما آیا همین دو حافظه برای یک کامپیوتر با توجه به وظایف آن‌ها کافی‌ست؟ به فایل‌های عکس خود که در کامپیوتر ذخیره کرده‌اید فکر کنید. آیا عکس یک برنامه است که بخواهد در حال اجرا باشد که همیشه در RAM حاضر باشد و یا آنقدر با اهمیت است که در کنار سیستم‌عامل در ROM ذخیره شود؟ یعنی هیچ گاه ممکن نیست این عکس تغییر کند و یا پاک شود؟

و یا به یکی از بازی‌های رایانه‌ای که چندین مرحله از آن را سپری کرده‌اید فکر کنید. شما وقتی دوباره به سراغ این بازی می‌روید، برنامه از ادامه‌ی جایی که بوده اجرا می‌شود. چطور CPU متوجه شد که باید برنامه را از کجا اجرا کند؟ البته جواب این بخش مشخص است: RAM چون برنامه اجرا شده و CPU دستورات لازم را دریافت کرده.

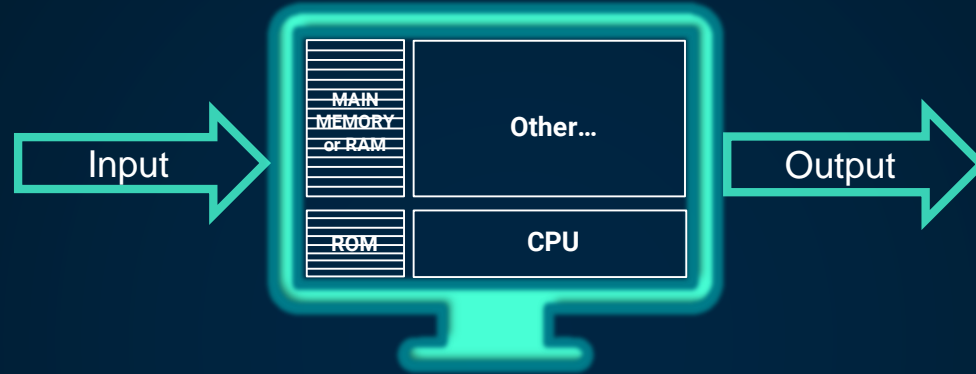
اما RAM همیشه این اطلاعات را در خود نگه می‌داشته تا شما بتوانید بازی را از جایی که قطع کردید ادامه دهید؟ پس در این صورت شما نمی‌توانستید تا رسیدن به آخرین مرحله‌ی بازی، کامپیوتر خود را خاموش کنید چون اطلاعات از داخل RAM پاک می‌شد.



با همین دو مثال متوجه می‌شویم که ما نیاز به یک (یا چند) حافظه‌ی جانبی داریم تا بتوانیم تمام اطلاعات مورد نیاز خود را در آن ذخیره کنیم. هرگاه به آن‌ها نیاز داشته باشیم، RAM اطلاعات را از حافظه‌ی جانبی دریافت می‌کند و به CPU می‌دهد تا درخواست شما به درستی اجرا شود.



# Input / Output

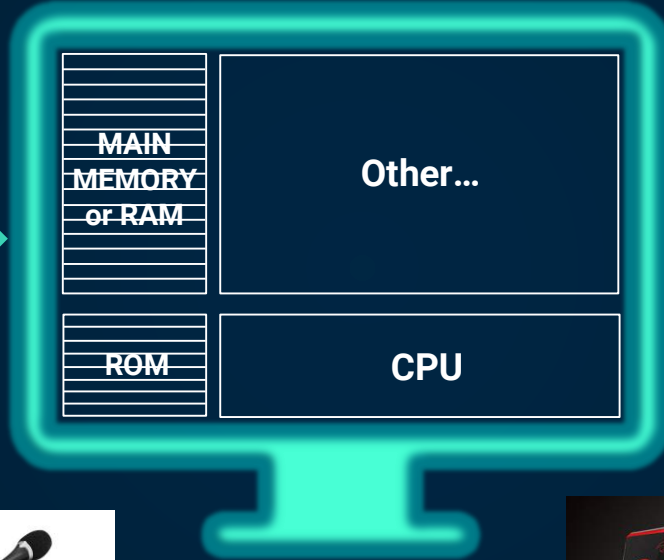


بخش‌هایی که تا اینجا بررسی کردیم همه مربوط به خود خود کامپیوتر بود و کاربر در بین آن‌ها نقشی نداشت. یکی از بخش‌های مهم دیگر وسایل ورودی و خروجی هستند که کاربران و برنامه نویسان و هرکسی که با کامپیوترها بر اساس نیاز خود کاری دارد، از طریق آن‌ها می‌تواند با کامپیوتر خود ارتباط برقرار کند.

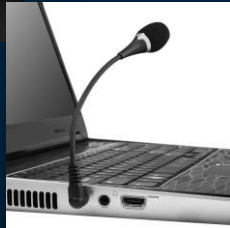
# Input / Output



Input



Output





تا اینجا کمی نسبت به طرز کار سخت افزار کامپیوتر دید پیدا کردید. در آینده در درس "معماری کامپیوتر" به طور کاملتری به این بخش‌ها آشنا خواهید شد.

در مرحله‌ی بعد، می‌خواهیم پا به دنیای برنامه‌نویسی گذاشته و درک کنیم برای تبدیل شدن به یک **برنامه‌نویس**<sup>۱</sup>، چه حداقل‌هایی را باید رعایت کنیم و از چه امکاناتی می‌توانیم استفاده کنیم.

برای شروع باید یک زبان برنامه‌نویسی را بلد باشیم...

1- developer or programmer

## یک زبان برنامه‌نویسی اصلاً یعنی چی؟

وقتی به جهان نگاه می‌کنید، تمام چیزهایی که به هر نحوی با یکدیگر در ارتباط باشند، طبق یک سری قانون و قاعده (پروتکل<sup>۱</sup>) این ارتباط را شکل می‌دهند. در ریز بینانه‌ترین حالت، این پروتکل‌ها تحت قوانین فیزیک تعریف می‌شوند و در دید بزرگ‌تر، با چیزهایی مثل زبان‌های طبیعی (فارسی، انگلیسی، عربی، الخ) سروکار دارند.

طبق همین روند، برای برقراری ارتباط با کامپیوترها و ارسال دستورالعمل به آن‌ها، نیاز به یک سری پروتکل و زبان خاص داریم.

در پایین‌ترین سطح، کامپیوترها فقط توانایی ذخیره‌ی ۰ ها و ۱ ها را دارند. می‌توانید این‌طور تصور کنید که کامپیوتر پر است از یک سری **سوییچ** (کلید روشن/خاموش) خیلی ریز که حالت‌های خاموش یا روشن دارند و از این‌ها به عنوان حافظه‌ی کامپیوتر استفاده می‌شود.



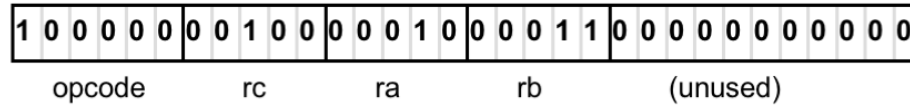
پس، دستوراتی که به یک کامپیوتر می‌دهیم (مثلا ۳ را با ۵ جمع کن) و داده‌هایی که می‌خواهیم برایمان ذخیره کند (مثلا قسمت مورد علاقه‌تان از سریال برکینگ بد :دی) همگی به همین شکل ذخیره می‌شوند، که کامپیوتر به وسیله‌ی یک سری بخش‌ها یا سطوح نرم‌افزاری و سخت‌افزاری، این‌ها را جوری تعبیر می‌کند که بتواند فرکانس‌های مختلف صدا را از اسپیکر پخش کند یا پیکسل‌های صفحه‌ی مانیتور را بر حسب نیاز تغییر دهد و ....



در حال حاضر لازم نیست درگیر شوید که کامپیوتر چطور این کار را انجام می‌دهد. شما قرار است در طول دوران تحصیل خود مرحله به مرحله با این موضوعات بیشتر آشنا شوید که به این مراحل در علم کامپیوتر سطوح **abstraction** گفته می‌شود. صرفا در حد یک درک اولیه عکس صفحه‌ی بعد را ببینید...

# Programming Languages

32-bit (4-byte) ADD instruction:



Means, to the BETA,  $\text{Reg}[4] \leftarrow \text{Reg}[2] + \text{Reg}[3]$

We'd rather write in *assembly language*:

`ADD(R2, R3, R4)`

Today

or better yet a *high-level language*:

`a = b + c;`

Coming up

نترسید! این یه جمع خیلی ساده‌ست که توی سه سطح تجرید<sup>۱</sup> توضیح داده شده.





هرچقدر که یک زبان برنامه‌نویسی به زبان خود ما نزدیک‌تر باشد، آن زبان سطح بالاتری دارد. ما در درس مبانی برنامه‌نویسی با بالاترین سطح زبان (یعنی چیزی شبیه به بخش نارنجی رنگ داخل شکل) کار داریم.



در بخش آبی رنگ نمونه‌ای از زبان اسمبلی را می‌بینید که یک نوع زبان برنامه‌نویسی سطح پایین است. زبان‌های سطح پایین عمدتاً به زبان ۰ و ۱ نزدیک‌ترند. رابط بین زبان‌های برنامه‌نویسی سطح بالا و سطح پایین چیزی به اسم **کامپایلر** است که یک برنامه به زبان سطح بالا را به یک برنامه‌ی سطح پایین ترجمه می‌کند.

توضیحات بیش‌تر درباره‌ی زبان اسمبلی و جزئیاتش را در درس "ریزپردازنده و زبان اسمبلی" یاد خواهید گرفت.



تا اینجا با وجود نزدیک‌تر شدن به دنیای  $\circ$  و  $1$  ها در این سطح هنوز کاملاً نمی‌توانیم چنین دستوری را به کامپیوتر بفهمانیم پس باید یک سطح پایین‌تر رفته و این دستورات را در قالب  $\circ$  و  $1$  به کمک سخت‌افزار کامپیوتر اجرا کنیم.



این کار توسط همان ثبات‌هایی که اول دستورکار توضیح داده شد انجام می‌شود... همانطور که در شکل هم دیدید، برای اجرا کردن عملیات جمع یک دستور شامل تعداد زیادی  $\circ$  و  $1$ ، اما کاملاً با معنی توسط CPU اجرا خواهد شد. در این دستور گفته شده که محتوای ثبات‌های  $ra$  و  $rb$  که به ترتیب ثبات‌های شماره‌ی  $2$  و  $3$  هستند به ALU داده شود و حاصل جمع آن‌ها که خروجی ALU است در ثبات  $rc$  ذخیره شود.



این‌که دقیقاً چطور این کارها انجام می‌شود و CPU چطور چنین برداشتی از روی چند تا  $\circ$  و  $1$  دارد را به طور کامل در درس "معماری کامپیوتر" فرا خواهید گرفت.

## الان در چه وضعیتی هستید؟



چون تا اینجا، کارگاه بیشتر شبیه کلاس درس بوده پیشنهادمون برای تغییر وضعیت اینه که حالا یکم شما برامون صحبت کنین. مثلا بگین چقدر با برنامه نویسی آشنایی دارین؟ تا حالا با زبان C برنامه ای نوشتین؟ راجع به این زبان چیزی می دونین؟ به نظرتون چرا مبانی برنامه نویسی رو با این زبان شروع کردیم و زبانی مثل پایتون برای شروع انتخاب نشده؟

## یه کم تاریخچه

اولین زبان برنامه‌نویسی سطح بالا که طراحی شد، زبانی به اسم **Plankalkül** ( از آلمانی: plan=برنامه‌ریختن + kalkul=محاسبه‌کردن) بود که در سال ۱۹۴۸ معرفی شد و تقریباً هیچ‌کس درباره‌ش چیزی نشنیده! این زبان صرفاً یک طراحی انتزاعی بود که تا ۳۰ سال بعد از معرفی شدنش هیچ کامپایلری برای آن معرفی نشد. اما نکته‌ی مهم این است که طراحی این زبان منجر به طراحی زبان‌های خانواده‌ی ALGOL شد.

زبان **ALGOL**<sup>۱</sup> و نسخه‌های مختلفش توسط ایده‌ها و طراحی مستقیم افراد بزرگی در حیطه‌ی علوم کامپیوتر، از جمله John McCarthy، John Backus، Peter Naur و Donald Knuth ساخته شدن و اولین نسخه‌ی این خانواده، یعنی ALGOL 58 در سال ۱۹۵۸ تولید شد که خیلی زبان کاربردی‌ای نبود. بعداً از روی ALGOL 58 زبان بسیار کاربردی (در زمان خودش البته!) ALGOL 60 را ساختند. بعد از کلی طراحی‌های جدید که با آزمون و خطای بسیار پیاده‌سازی شدند، نهایتاً زبان برنامه‌نویسی **B** توسط Ken Thompson و Dennis Ritchie ساخته شد.

```
void continue_last_game(FILE * fp_user, char name_game[]); void decide_for_saving(FILE * fp_user, char name_game[]); struct node * create_node(FILE * FP, char C[], int i); struct node * create_list(); struct node * random(struct node * list); struct node * check_list(struct node * list); int main(int argc, char const *argv[]){ printf("Please enter your name: "); struct game user_name; scanf("%s", user_name.name); int save_res = save_name(user_name.name);
```

بعد از گذشت مدتی خود طراحان زبان B به نتیجه رسیدند که به زبان کامل‌تری نیاز دارند و زبان C را خلق کردند. البته از آن‌جا که هنگام طراحی این زبان می‌خواستند هم‌چنان با زبان B تطابق داشته باشد، در نتیجه این زبان هم ایراداتی دارد که در آینده خودتان متوجه آن‌ها خواهید شد. شاید برایتان جالب باشد که زبان B و C داریم، ولی زبان A نداریم. (البته اگر بخواهید می‌توانید ALGOL را به عنوان زبان A در نظر بگیرید).

```
int i, a; if(save_res == 0){ printf("Hello %s :)\n", user_name.name); printf("Welcome to your first game...\n"); printf("You can quit the game by pressing 0\n"); make_new_game(&user_name); } else if(save_res == 1){ printf("Hello %s :)\n", user_name.name); printf("Which one do you like?\n"); printf("1) Play a new game\n2) Continue your last game\n"); scanf("%d", &a); if(a == 1){ printf("Welcome to your new game...\n"); printf("You can quit the game by pressing 0\n"); make_new_game(&user_name); }
```



بعد از پیدایش زبان C و محبوبیت بی‌اندازه‌ش به دلیل استفاده شدن در **هسته‌ی سیستم‌عامل Unix** (که بعداً شد Linux)، این زبان به استاندارد زبان‌های **imperative**<sup>۱</sup> تبدیل شد و سینتکسش<sup>۲</sup> (Syntax) تبدیل به استانداردی شد که اکثر زبان‌های برنامه‌نویسی‌ای که امروزه از آن‌ها استفاده می‌شود از این زبان الگو گرفته‌اند. (جز زبان‌هایی که بر اساس Lisp نوشته شدن و جلوتر به کم درباره‌شون حرف می‌زنیم.) این زبان استانداردهای مختلفی دارد که هر کدام نام مختص به خود را دارند. معروف‌ترین آن‌ها C99 (سال ۱۹۹۹ استاندارد شد) و جدیدترین‌شان C18 (سال ۲۰۱۸ استاندارد شد) هستند. شما با توجه به محیطی که برای برنامه‌نویسی خود انتخاب می‌کنید، احتمالاً با یکی از این دو ورژن سر و کار خواهید داشت (اما نگران نباشید! تفاوت بین آن‌ها خیلی زیاد نیست؛ طوری که برای برنامه‌نویس‌های تازه‌کار، تقریباً اصلاً به چشم نمی‌آید).

۱- یعنی زبان‌هایی که می‌توانیم در آن‌ها مقدار یک متغیر را تغییر بدیم.

۲- یعنی قوانین مربوط به چگونگی نوشتن دستورهای یک زبان

# شما یک کاربر هستید

هر کدام از شما بخشی از زمان را با گوشی همراه، لپ تاپ، کامپیوترهای شخصی و ... سپری می‌کنید، پس یک کاربر هستید.

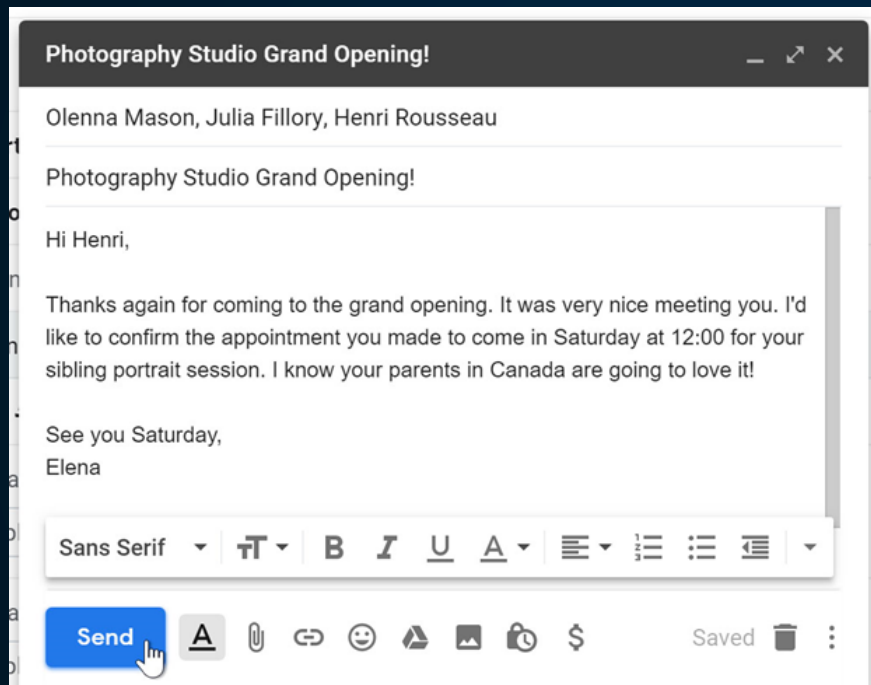
شما به عنوان یک کاربر نیاز دارید که کار با چند ابزار و برنامه را حتما بلد باشید تا بتوانید کارهای مورد نیاز خود را مثل کارهای مربوط به دانشگاه و غیره را انجام دهید.

## توجه توجه توجه تو...تو...

می‌خواهم با بچه‌هایی صحبت کنم که با دیدن کلی مطالب ممکنه استرس بگیرن و فکر کنن که وای چقدر چیز باید یاد بگیرم و چقدر عقبم یا اینکه بگن مشکل از منه و هزار جور صحبت دیگه از این دست...

بچه‌ها هدف این کارگاه این نیست که بگه شما باید تا آخر این ترم همه چیز رو یاد گرفته باشید. تمام سایت‌هایی که در اسلایدهای بعدی معرفی شده، تمام نرم‌افزارها و غیره و غیره همه‌شون رو شما یاد خواهید گرفت حتی اگر هیچی ازشون ندونین. توی این دستورکار سعی شده تعداد زیادی از این منابع مهم برای یک دانشجوی مهندسی کامپیوتر یک جا جمع بشه و یک کمکی هم باشه برای علاقه‌مندای به یادگیری که دوست دارن خودشون برن و بیشتر از درس مطالبی رو دنبال کنن. امیدواریم از خوندن دستورکار لذت ببرید.

# گوگل



ایمیل یکی از راه‌های ارتباطی رسمی امروزه هست که سایت‌هایی مثل google و yahoo به طور رایگان این امکان را ارائه می‌دهند. محتوای هر ایمیل رسمی‌ای که می‌نویسیم از بخش‌های مختلفی تشکیل شده است. به کمک استاد کارگاه بخش‌های مختلف ایمیل زیر را بررسی کنید و مروری نیز بر نحوه‌ی فرستادن ایمیل داشته باشید.

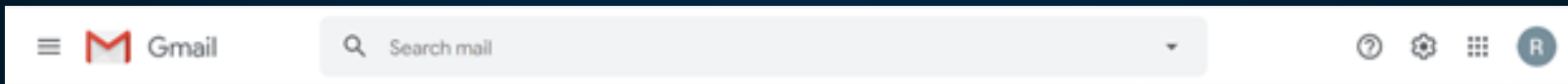




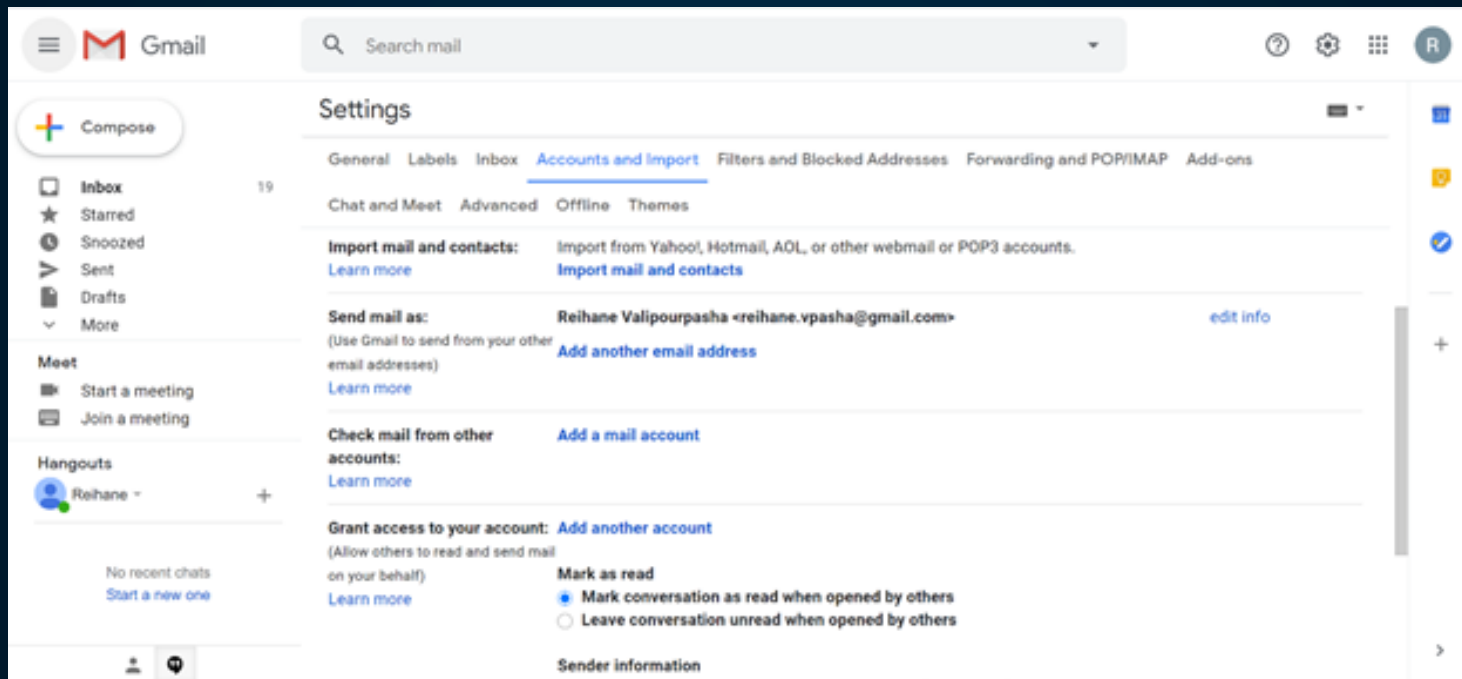
همه‌ی ما بعد از ورود به دانشگاه یک حساب ایمیل دانشگاه داریم که رسمی‌تر از حساب‌هایی مثل gmail هست و نشان‌دهنده‌ی این است که ما دانشجوی دانشگاه هستیم.

یکی از کارهایی که برای داشتن دسترسی کامل به همه‌ی ایمیل‌ها به صورت یک‌جا می‌توانیم انجام دهیم این است که حساب ایمیل دانشگاهمان را به حساب gmail خود اضافه کنیم. در ادامه روش انجام این کار را مشاهده می‌کنید:

اول از همه اکانت gmail خود را باز کنید و در قسمت setting، یعنی عکس زیر دکمه‌ی see all setting را انتخاب کنید.



بعد دقیقا از قسمت check mail for other accounts مانند شکل زیر add a mail account را انتخاب کنید.



سپس آدرس ایمیل دانشگاه خود را وارد کنید و اطلاعات خواسته شده را هم کامل کنید.





در نهایت شما موفق شدید! حالا به پیام‌هایی که به ایمیل دانشگاه شما ارسال می‌شود از طریق حساب google account خود دسترسی دارید.



لینک زیر هم شامل دستورالعمل سایت Gmail Help برای انجام این کار با سیستم‌های مختلف است که می‌توانید در صورت نیاز به آن مراجعه کنید.



Gmail help

<https://b2n.ir/137395>



بعد از تبریک دوباره برای ورود به دانشکده، ما اینجا هستیم تا حقایق رو برای شما آشکار کنیم و اولین حقیقت این هست که شما در این دانشکده لقب سرخپوستی "بسیار تمرین آپلود کننده" رو از همین هفته‌های اول تا فارغ‌التحصیلی به دوش می‌کشین.

اما فقط آپلود شدن تمرین‌ها مهم نیست. این هم مهمه که چجوری آپلود بشن.

برای اینکه تمرین‌ها رو آپلود کنیم اولین قدم لازمه که ببینیم چی ازمون خواسته شده؟! مثلاً یک فایل pdf یا چند فایل کد و گزارش‌های مربوط به اون...

## فشرده‌سازی و zip

اگر بخواهیم چند فایل را به عنوان تمرین تحویل دهیم، برای منظم‌تر شدن محتوایی که قرار هست آپلود شود و یا گاهی به این دلیل که تنها راه فشرده کردن فایل‌هاست، باید از ابزارهای فشرده‌سازی مثل zip استفاده کنیم که روش انجام این کار را در لینک زیر با هم می‌بینیم:



Making ZIP

<https://b2n.ir/771448>

در بیشتر مواقع لازم است تا گزارشی که نوشتیم را به فرمت pdf تبدیل کنیم. در این صورت اگر فایل word باشد، به راحتی می‌توانیم هنگام save کردن فرمت را pdf قرار دهیم و ذخیره کنیم. اما اگر دست‌نوشته باشد می‌توانیم از برنامه‌هایی مثل CamScanner استفاده کنیم تا عکس‌های ما از نوشته‌ها را به ترتیب و به صورت یک‌جا در یک فایل pdf قرار دهد.

## سایت های مفید

برای ما دانشجویهای مهندسی کامپیوتر، دسترسی به اینترنت مثل اکسیژن حیاتی و در کنارش استفاده از بعضی سایت ها گاهی از نون شب هم برامون واجبتر می شه :) حالا تعدادی از این سایت ها رو با هم می بینیم و از شما می خواهیم که حتما بعد از کلاس، دریای بی کران اطلاعات اون ها بیش تر بررسی کنین. منبع لایتناهی برای یادگیری هر چیزی که بخواین:



W3Schools

<https://www.w3schools.com/>



Coursera

<https://www.coursera.org/>



Udemy

<https://www.udemy.com/>



Tutorials Point

<https://www.tutorialspoint.com/index.html>



Geeks for Geeks

<https://www.geeksforgeeks.org/>

هر سوالی داری قبلاً یکی اینجا پرسیده و جواب گرفته: 😊



Stack Overflow

<https://stackoverflow.com/>

بعداً یاد می‌گیرین که چجوری کدهاتون رو اینجا نگه دارین: 😊



Gitlab

<https://about.gitlab.com>



Github

<https://github.com/>

و در نهایت مهم‌ترین این لیست گوگل و متعلقات اون هست:  
Google و Google Docs و Google Sheets و ... 😊

# آشنایی با نرم افزارهای office

پرکاربردترین نرم افزارها از لیست بلند بالای office می‌شه به word و excel و powerpoint اشاره کنیم که قراره خیلی ازشون استفاده کنیم. 😊

از لینک زیر می‌تونیم این مهارت‌ها رو یاد بگیریم:



Office Tutorial

<https://b2n.ir/182501>



## معرفی سایت تمرین تایپ

تایپ سریع و اصولی یکی از مهارت‌هایی هست که بهتره ما به عنوان دانشجوی مهندسی کامپیوتر بلد باشیم چون خواه ناخواه زمان زیادی از روز و در ادامه زمان زیادی از عمرمون رو پشت کامپیوتر و لپ‌تاپ نشستیم و باید مراقب این اصول باشیم تا هم از این نشستن آسیب نبینیم و هم سرعت و بهره‌وری مون زیاد بشه.

سایت زیر منبع خوبی برای یادگیری و بعد تمرین این مهارت است:



Typing Club

<https://www.typingclub.com/>

# یه دیکشنری مختصر



توی رشته‌ی ما، یه سری اصطلاح رو خیلی زیاد باهاشون مواجه می‌شید. برای این‌که خیلی گیج نشید و به تصور اولیه ازشون داشته‌باشید، اونا رو این زیر براتون توضیح می‌دیم:

## سیستم عامل

ترجمه‌ی عبارت Operating System (یا به اختصار، OS) هست. سیستم‌عامل برنامه‌ایه که دستورات Load شدنش (که بهش می‌گن Firmware) توی ROM کامپیوتر ذخیره می‌شه که باعث می‌شه موقع روشن شدن کامپیوتر، اولین برنامه‌ای باشه که اجرا می‌شه. وظیفه‌ی اصلی سیستم‌عامل مدیریت کردن تعداد زیادی برنامه به صورت موازی و ایجاد هماهنگی بین اونا برنامه‌هاست. سیستم‌عامل‌ها انواع مختلف و ورژن‌های مختلف دارن. یکی از این انواع کمتر شناخته‌شده بین تازه‌کارها، سیستم عامل Linux هست که توزیع‌های بسیار زیادی داره، از جمله Ubuntu, Fedora, CentOS, ArchLinux و... که ویژگی مشترک بین این‌ها هسته یا کرنل لینوکسه. کرنل یک سیستم‌عامل کدیه که قسمت‌های اصلی سیستم‌عامل رو توی خودش داره و در مورد Linux به زبان C نوشته شده. (در مورد این بحث بیش‌تر در درس سیستم‌عامل می‌خونید)

## زبان‌های برنامه‌نویسی

در مورد خودشون توضیح دادیم، اما خوبه که بدونید چندین پارادایم (می‌شه گفت طرز تفکر) مختلف در زبان‌های برنامه‌نویسی وجود دارن که یه توضیح مختصر و مثال از بعضی‌هاشون میاریم. تفاوت بین این طرز تفکر ها نحوه‌ای هست که مسائل برنامه‌نویسی رو به قسمت‌های کوچیک‌تر می‌شکنن.

## برنامه‌نویسی procedural

این تفکر یه مسئله رو به متغیرها، ساختمان‌های داده (توی درس ساختمان داده و الگوریتم، به اختصار DS می‌خونید) و توابع می‌شکنه و برنامه‌هاش صرفاً از پشت هم اجرا شدن یک سری تابع (function, method, ) (procedure) تشکیل شدن. این توابع صرفاً یک‌سری دستور هستن که یک‌بار نوشته می‌شن و چندین بار و با پارامترهای مختلف اجرا می‌شن. (جلوتر توی همین درس می‌خونید درباره‌شون) که این زبان‌ها همگی imperative هستن. زبان‌های C، Fortran و Pascal جزو این دسته هستن.

## برنامه‌نویسی Object-Oriented یا OOP

این تفکر مسئله رو به قسمت‌هایی به اسم Object می‌شکنه که این Object ها مجموعه‌ای از توابع و متغیرها در یک بسته هستند و برنامه‌های این زبان از ارتباط این بسته‌های مختلف با هم‌دیگه تشکیل می‌شن. (توی درس برنامه‌نویسی پیش‌رفته با این بحث بیشتر آشنا می‌شید). زبان‌هایی مثل ++C و Java و #C جزو این دسته هستند.

## برنامه‌نویسی Functional

تفکری هست که تنها استفاده از مقدارهای ثابت (constant) رو مجاز می‌دونه، هیچ مقداری قابل تغییر نیست و ایجاد مقدارهای جدید تنها با ایجاد یک constant جدید ممکن هست. دلیلش هم این هست که با این تفکر می‌شه بهتر درستی عملکرد یک برنامه رو اثبات کرد. (درباره‌ی این پارادایم بیشتر در درس «زبان‌های برنامه‌نویسی» می‌خونید) زبان‌هایی مثل Lisp و Haskell و ML جزو این دسته هستند. و بسیار پارادایم‌های دیگه هم وجود دارن که در این‌جا فرصت نمی‌کنیم بیاریم‌شون ولی پیشنهاد می‌کنیم حتما از این صفحه‌ی ویکی‌پدیا به مطالعه‌ی روشون داشته‌باشید:



## پایگاه داده یا Database (به اختصار: DB)

مجموعه‌ای از داده‌ها هستند که با نظم خاص و با استفاده از تکنیک‌های مختلفی کنار هم چیده می‌شن. استفاده از Database ها به این دلیل هست که دسترسی‌مون به داده‌هایی که برای برنامه‌هامون نیاز داریم سریع‌تر بشه و یک سطح Abstraction رو وارد کارمون با دیتا بکنیم. (مثلا به جای این‌که بگیم برو فایل فلان رو بخون و خطی که کاراکتر اولش عدد ۳ هست رو بردار، به دیتابیس می‌گیم داده‌ای که idش شماره ۳ هست رو برای من بفرست.)

## محیط توسعه یا Integrated Development Environment یا IDE

برنامه‌هایی هستند که با کنار هم آوردن چندین و چند ابزار کنار هم، روند توسعه‌ی برنامه رو برای برنامه‌نویس آسون می‌کنن. این‌ها با یک ویرایش‌گر متن ساده کار خودشون رو شروع می‌کنن و ابزارهایی مثل ( Linter برای مرتب کردن کد)، Compiler، ابزارهای کنترل ورژن (مثل Git که جلوتر می‌بینیدش)، Debugger (برای رفع ایراد کد)، Syntax Highlighter (برای عوض کردن رنگ قسمت‌های مختلف کد) و خیلی آپشن‌های دیگه رو به اون ویرایش‌گر اضافه می‌کنن. محیط‌های توسعه‌ی زیادی داریم که اون‌ها رو این زیر لیست کردیم:

محصولات شرکت JetBrains مثل CLion برای زبان C و IntelliJ برای زبان Java و PyCharm برای زبان Python.

## ویژوال استودیو (Visual Studio)

برای توسعه‌ی زبان‌های بسیار زیادی ازش می‌تونه استفاده بشه ولی استفاده‌ی اصلیش برای زبان‌های توسعه‌یافته‌شده توسط مایکروسافت مثل #C و ++C Microsoft هست. ویژوال استودیو کد (VS Code) که در ابتدا یک ویرایش‌گر متن ساده هست ولی به سرعت با استفاده از ابزارهایش که به صورت Open-Source منتشر شدن تبدیل به یک IDE می‌شه و در سال‌های اخیر هم به شدت مورد مقبولیت قرار گرفته. ساب‌لایم (Sublime Text) و ++Notepad که باز ویرایش‌گرهای متن هستن ولی می‌تونن قابلیت‌های دیگه‌ای هم به صورت محدود خودشون اضافه کنن.

## سرور - کلاینت

سرور به طور خیلی خلاصه، کد یا کامپیوتری هست که با گرفتن درخواست‌های کلاینت‌ها (کاربران) پردازش‌هایی روی درخواست‌هاشون انجام می‌ده و جواب رو بهشون برمی‌گردونه. مثال خیلی ساده‌ی سرور می‌شه سایت گوگل که درخواست سرچ شما (کلاینت) رو می‌گیره و به پاسخی بهتون برمی‌گردونه.

## بک‌اند - فرانت‌اند

خیلی از برنامه‌ها از دو قسمت تشکیل شدن. یک قسمت که کاربر باهاش ارتباط برقرار می‌کنه و بهش می‌گن Front-End یا User Interface = UI و یک قسمت که پردازش رو روی داده‌های کاربر انجام می‌ده و بهش می‌گن Back-End که این قسمت از دید کاربر محفوظ هست. (این اصطلاح‌ها رو اکثراً در حوزه‌ی برنامه‌نویسی موبایل یا برنامه‌نویسی وب خواهید دید.)

;