

آینده زمان نیست، بلکه طرز فکر ماست؛ برای تغییر آینده باید  
طرز فکرمان را تغییر دهیم.

# مقدمه بر آموزش برنامه‌نویسی هوش مصنوعی

گردآوری و تالیف

محمد شعبی

۱۳۹۹

مؤسسه آموزشی تألیفی ارشدان

پیشگفتار ناشر:

به نام ایزد دانا که آغاز و انجام از آن اوست

هرگز دل من زعلم محروم نشد      کم ماند زاسرار که مفهوم نشد  
اکنون که به چشم عقل در می نگرم      معلومم شد که هیچ معلوم نشد

ای دانای بی همتا، ای بخشنده ایی که ناخواسته عطا فرمایی و هر نیازمندی را به عدالت بی نیاز گردانی، مگر اینکه نالایق باشد و آن عنایت را به باژگونه از دست دهد. در عرصه پیشرفت تکنولوژی در هزاره سوم، هنوز نیاز بر مطالعه کتاب در کنار استفاده از منابع کامپیوتری و اینترنت احساس می شود. از این بابت خوشحالیم که می توانیم در جهت اعتلای علم، دانش و فرهنگ کشور قدمی هر چند کوچک برداریم.

و من الله التوفیق

دکتر شمس الدین یوسفیان

مدیر مسئول انتشارات ارشدان



## مقدمه

مسلماً یکی از سودمندترین اختراعات بشر تا به امروز، کامپیوترها بوده‌اند. دستگاه‌هایی که هزاران بار سریعتر از انسان فکر می‌کنند و سرعت عملشان به طرز غیر قابل باوری بالاست. سرعت و قدرت این وسایل، امکان انجام خیلی از کارهایی را فراهم کرده است که انسان به طور عادی از انجام آن‌ها عاجز بود.

اما با وجود این مزایا، یک مشکل وجود داشت: این دستگاه‌ها به همان اندازه که قوی بودند، کم هوش هم بودند؛ آن‌ها به طور عادی هیچ عملی انجام نمی‌دادند مگر این که به صراحت از آن‌ها درخواست می‌شد. اما این درخواست چگونه باید صورت می‌گرفت؟ باید راهی برای گفتگو با آن‌ها پیدا می‌شد. و در این زمان بود که متخصصان تصمیم گرفتند زبان‌های مخصوصی را بوجود بیاورند تا بتوانند با کامپیوترها گفتگو کنند. این زبان‌های مخصوص به اصطلاح **زبان‌های برنامه نویسی کامپیوتر** نام گرفتند. به نسل اولیه زبان‌های برنامه نویسی، زبان‌های سطح پایین گفته می‌شد، چرا که به سختی قابل یادگیری و به

کارگیری بودند. پیاده سازی اعمالی ابتدایی توسط این زبان‌ها باعث می‌شد مدت‌ها وقت با ارزش برنامه نویسان گرفته شود. اما وضع به همین منوال نماند و با گذشت زمان زبان‌های جدیدی به وجود آمدند که آموختن آن‌ها راحت‌تر بود. طی سه دهه‌ی اخیر، صنعت کامپیوتر مورد هجوم انواع زبان‌های برنامه نویسی قرار گرفته است.

سیستم‌های کامپیوتری جدید، تاثیرات وسیع، و رشد یابنده‌ای بر اکثر فعالیت‌های بشری داشته و دارند. کامپیوتر به ما این امکان را داده است تا زمینه‌های جدیدی از تحقیقات در علوم ایجاد کنیم؛ پیش‌تر، به سبب کمبود داده‌ها و محدودیت در انجام تحلیل‌ها و محاسبات عددی، علوم

چندان شناخته شده نبودند. کامپیوتر، پیشرفت‌های تکنولوژی، را تسهیل کرده و بعنوان وسیله‌ای برای کنترل فرایندهای صنعتی، به گستردگی مورد استفاده قرار می‌گیرد. اکثر سیستم‌های حسابداری و بانکی، اینک کامپیوتری شده و در فعالیت‌هایی مثل مدیریت موجودی و انبار، پرداخت حقوق حمل و نقل و مراسلات، از کامپیوتر در حد وسیع استفاده می‌شود. سازمان‌های دولتی، اینک برای ذخیره و بازیابی اطلاعات، کامپیوتر را بکار می‌گیرند. در دانشگاه‌ها برای ذخیره و بازیابی اطلاعات، امور حسابداری و پرداخت حقوق، برنامه ریزی دروس و ثبت نام دانشجویان و فعالیت‌های دیگر از کامپیوتر بهره برداری می‌شود. بسیاری از سیستم‌های کتابداری، اینک کامپیوتری شده‌اند و در کتابخانه‌ها، حتی برای نگاهداری و بازیابی اسناد و مدارک و چکیده‌های علمی، از کامپیوتر استفاده می‌کنند. کامپیوتر در تمام فعالیت‌هایی که در آنها، پردازش سریع حجم زیادی از اطلاعات، مورد نیاز باشد، بکار برده می‌شود. هر گونه علامت گذاری به منظور توصیف الگوریتم‌ها و ساختمان داده‌ها می‌تواند یک زبان برنامه نویسی محسوب شود. دنیای برنامه نویسی دنیای بسیار جالبی است که نیاز به وقت و قدرت تجزیه و تحلیل بالایی دارد، زیرا شما باید مسائل را به خوبی تجزیه و تحلیل کنید و بعد برای آن مسئله راه حلی بیابید. حال بهتر است که وقت را غنیمت بشماریم و هر چه سریعتر به سراغ آشنایی با این زبان‌های برنامه نویسی برویم.

در این کتاب می‌خواهیم در رابطه با دو زبان برنامه نویسی هوش مصنوعی صحبت کنیم. LISP و PROLOG نام دوزبان برنامه نویسی هوش مصنوعی می‌باشد که امروزه بسیار پرکاربرد است. این کتاب دارای ۶ بخش می‌باشد که بخش اول آن در رابطه با مقدمه زبان‌های برنامه نویسی صحبت شده است، در این بخش می‌توان مطالبی مانند روش‌های اجرای برنامه، معیارهای یک زبان خوب و دلایل مطالعه زبان‌های برنامه

نویسی مختلف، اطلاعاتی بدست آورد. در بخش دوم نیز می‌توانیم با سیر تکاملی زبان‌های برنامه نویسی، مثل نسل‌های برنامه نویسی و تقسیم بندی کلی زبان‌ها، دسته بندی آنها و مدل‌های برنامه نویسی صحبت شده است. در بخش سوم نیز مروری بر تاریخچه چند زبان برنامه نویسی شده است. بخش چهارم نیز برای آشنایی خواننده با تاریخچه هوش مصنوعی می‌باشد، در این بخش در رابطه با تست تورینگ نیز (که یکی از مهمترین مباحث هوش مصنوعی می‌باشد) صحبت شده است. در فصل پنجم می‌توانید با مقدمات زبان Lisp آشنا شوید و با روش‌های تعیین توابع، مقدار یابی و ساخت توابع نیز آشنا می‌شوید. هر کدام از این مباحث دارای مثال‌هایی در ارتباط با مطلب مورد نظر آورده شده است تا مطالب را برای خواننده روشن‌تر کند. در ابتدای بخش ششم، در رابطه با زبان‌های توصیفی و رویه‌ای (منطقی) صحبت شده است تا خواننده را با مطالبی که قرار است در این بخش آموزش ببیند آشنا کند. در ادامه این بخش می‌توانید با ویژگی‌های زبان Prolog و گزاره‌های خبری و شرطی، مسائل مرتبط با rule و fact، انواع داده‌ها، عمل تطبیق، معنای توصیفی و رویه‌ای مسائل در prolog، آشنایی با لیست، اپراتورها و عبارات محاسباتی و چند رابطه کاربردی دیگر آشنا شوید.

بهمن ۱۳۹۹



## فهرست مطالب

**بخش اول: مقدمه ای بر زبان های برنامه نویسی ..... ۱۳**

معماری های نرم افزار ..... ۱۳

محیط های عملیاتی و محیط میزبان ..... ۱۳

روش های اجرای برنامه ..... ۱۴

معیارهای یک زبان برنامه نویسی خوب ..... ۱۵

دلایل مطالعه زبان های برنامه نویسی مختلف ..... ۱۶

**بخش دوم: سیر تکاملی زبان های برنامه نویسی ..... ۱۹**

نسل های برنامه نویسی ..... ۱۹

تقسیم بندی کلی زبان های برنامه نویسی ..... ۲۱

دسته بندی زبان های برنامه نویسی ..... ۲۲

مدل های برنامه نویسی ..... ۲۴

**بخش سوم: مروری بر تاریخچه چند زبان برنامه نویسی معتبر ... ۲۵**

زبان برنامه نویسی C ..... ۲۵

زبان برنامه نویسی LOGO ..... ۲۵

زبان برنامه نویسی LISP ..... ۲۶

زبان برنامه نویسی ALGOL ..... ۲۶

زبان برنامه نویسی PASCAL ..... ۲۶

زبان برنامه نویسی ADA ..... ۲۷

زبان برنامه نویسی Basic ..... ۲۷

زبان برنامه نویسی COBLOL ..... ۲۹

زبان برنامه نویسی PL/1 ..... ۲۹

زبان برنامه نویسی FORTRAN ..... ۳۰



**بخش چهارم: هوش مصنوعی از چند دیدگاه مختلف ..... ۳۳**

مزیت کامپیوترهای هوشمند نسبت به انسان ..... ۳۵

مهمترین قابلیت های ربات هوشمند ..... ۳۵

تاریخچه هوش مصنوعی ..... ۳۶

تست تورینگ ..... ۳۶

عوامل موفقیت در تست تورینگ ..... ۳۸

مباحثی هوش مصنوعی ..... ۳۹

**بخش پنجم: زبان برنامه نویسی لیسپ ..... ۴۱**

تاریخچه زبان لیسپ ..... ۴۱

S - expression ..... ۴۲

مقداریابی در LISP ..... ۴۵

Interpreter ..... ۴۵

تابع LIST ..... ۴۶

تابع LENGTH ..... ۴۷

single quote ( ' ) ..... ۴۷

تابع CAR (FIRST / HEAD) ..... ۴۸

تابع CDR (REST / TAIL) ..... ۴۹

تابع CONS ..... ۴۹

تابع ATOM ..... ۵۲

تابع EQ ..... ۵۲

تابع عملیات شرطی COND ..... ۵۳

ساخت تابع به روش define و defun ..... ۵۵

تعریف تابع به روش define ..... ۵۵

تعریف تابع به روش defun ..... ۵۶

## بخش ششم: زبان برنامه نویسی پرولوگ..... ۵۷

ویژگی های زبان توصیفی ..... ۵۷

حل مسائل در هوش مصنوعی ..... ۵۸

ویژگی های زبان پرولوگ ..... ۵۹

گزاره های خبری و گزاره های شرطی ..... ۶۰

نوشتن درخواست در پرولوگ ..... ۶۱

اضافه کردن روابط دیگر ( قاعده جنسیت ) ..... ۶۲

دستور شرطی ( - : ) در پرولوگ ..... ۶۲

انواع داده در پرولوگ ..... ۶۴

قواعد کلی تطبیق ..... ۶۶

معنای توصیفی و رویه ای برنامه ها در پرولوگ ..... ۶۷

مجموعه ای از تمرین های توصیفی و رویه ای ..... ۶۹

انواع لیست در پرولوگ ..... ۷۳

رابطه Concat ..... ۷۴

رابطه sublist ..... ۷۵

رابطه subset ..... ۷۵

اپراتورها و عبارات محاسباتی در زبان ..... ۷۶

رابطه sumlist ..... ۷۶

Backtracking در زبان پرولوگ ..... ۷۷

عملگر cut (!) ..... ۷۷

مثال های کاربردی در زبان لیسپ و پرولوگ ..... ۷۸

بررسی n! در دو زبان LISP و PROLOG ..... ۷۸

چگونه زبان لیسپ را اجرا کنیم؟ ..... ۸۰

چگونه زبان پرولوگ را اجرا کنیم؟ ..... ۸۱



# بخش اول

## مقدمه ای بر زبان های برنامه نویسی

### معماری های نرم افزار

۱. دوره کامپیوترهای بزرگ مانند:  
محیط دسته‌ای و محیط محاوره‌ای
۲. دوره کامپیوترهای شخصی مانند:  
کامپیوترهای شخصی و سیستم‌های جاسازی شده
۳. دوره سیستم‌های توزیع شده و اینترنت مانند:  
هر یک از دوره‌های معماری نرم افزار بالا باعث ساخت نرم افزارهای خاصی شد، در دوره شبکه باعث ایجاد زبان‌های برنامه نویسی Client/server و HTML و... شد؛ و یا در دوره کامپیوترهای شخصی ما شاهد ساخت نرم افزارهای گرافیکی بودیم و هرکدام باعث پیشرفت روزافزون علم برنامه نویسی بودیم.

### محیط‌های عملیاتی و محیط میزبان

محیط میزبان، محیطی است که برنامه در آن ایجاد، تست و اشکال زدایی می‌شود؛ محیط عملیاتی، محیطی است که برنامه بر روی آن اجرا می‌شود.

برای مثال برنامه‌ای جهت اجرا بر روی یک کامپیوتر شخصی، در ابر کامپیوتر ساخته می‌شود و بدین ترتیب، ابر کامپیوتر به عنوان محیط میزبان و کامپیوتر شخصی به عنوان محیط عملیاتی شناخته می‌شود.

## روش‌های اجرای برنامه

۱. به صورت ترجمه یا کامپایل: در این روش دستورات برنامه به زبان ماشین کامپیوتر مورد نظر، تبدیل شده و این فایل در هر زمان بر روی کامپیوتر قابل اجرا خواهد بود. مانند زبان‌هایی مثل C، ++C، پاسکال و...

۲. به صورت تفسیر: در این روش دستورات برنامه خط به خط تفسیر شده و در همان لحظه اجرا می‌شوند. مانند زبان‌هایی مثل HTML, PROLOG, PERL, BASIC.

۳. به صورت ترکیبی از ترجمه و تفسیر: در این روش ابتدا یک ترجمه مختصری بر روی برنامه مبدأ انجام می‌شود و به کد میانی تبدیل می‌شود، سپس کد میانی با سرعت بیشتری تفسیر و اجرا می‌شود.

مزایا و معایب تفسیر و ترجمه: ترجمه باعث می‌شود تا اطلاعات موجود در برنامه در لحظه اجرا از بین برود که این خود، یکی از ضعف‌های متد ترجمه است، و همچنین انعطاف پذیری کم آن نسبت به روش تفسیر است. در روش تفسیر نیز به تعداد دفعات مورد نیاز جهت اجرای دستورات، تفسیر صورت می‌گیرد؛ برای نمونه یک حلقه که که تعداد دفعات چرخش آن ۱۰۰ بار است، دستورات داخل آن نیز باید ۱۰۰ بار تفسیر و ۱۰۰ بار اجرا شوند. در تفسیر تمام شاخه اجرایی برنامه از لحاظ عدم وجود خطا بررسی نمی‌شوند، لذا امکان بروز خطا در لحظه

اجرا، در آن بیشتر است. حجم برنامه‌های اجرایی به روش تفسیر کم است! در روش فسیر نیز مفسر همیشه باید برای اجرای برنامه حضور داشته باشد. همچنین سرعت اجرای برنامه به روش تفسیر بسیار کند است.

ابزارهای مورد نیاز جهت ترجمه: هریک از این ابزارها نیز، خود می‌توانند به عنوان یک مترجم دسته بندی شوند: کامپایلر، اسمبلر، بارکننده، متصل کننده، پیش پردازنده

### معیارهای یک زبان برنامه نویسی خوب

۱. وضوح، سادگی و یکپارچگی: این معیار بدین معناست که برنامه تولیدی ما باید از حداقل تعداد مفاهیم برخوردار باشد و همچنین دارای قوانینی ساده باشد تا بتوان آنها را به سادگی ترکیب کرد. در کنار این مزایا این برنامه باید دارای قابلیت خواندن و نوشتن بالایی هم باشد.
۲. تعامد: تعامد به معنی امکان ترکیب ویژگی‌های زبان‌های مختلف با یکدیگر است.
۳. طبیعی بودن برای کاربر: این معیار به معنای راحتی آن برنامه در برابر دیگر زبان‌هاست و این راحتی باید قابل درک باشد. در کل هر زبان باید هنگام استفاده در کاربردهای خاص خود، مناسب به نظر آید.
۴. پشتیبانی از تجرد: به معنی امکان ایجاد ساختارهای داده‌ای جدید است.
۵. هزینه‌های استفاده:

- هزینه اجرا: به معنی میزان استفاده از منابع نظیر زمان و حافظه است.
  - هزینه ترجمه: به معنای زمان کامپایل برنامه می‌باشد.
  - هزینه ایجاد و آزمایش برنامه: به معنای این است که باید برنامه مرد نظر را در کمترین زمان ممکن ایجاد و اشکال زدایی کرد.
  - هزینه نگهداری و پشتیبانی: به معنای این است که برنامه باید به سادگی در اثر بروز تغییرات احتمالی در آینده، خود را تغییر دهد و به روز رسانی شود.
۶. سادگی بررسی درستی برنامه:
- رسمی: به کمک روشهای ریاضی است.
  - غیر رسمی: به کمک روشهایی نظیر تست با ورودی‌های مختلف است.
۷. قابلیت حمل برنامه: این معیار به منای امکان استفاده از برنامه‌های آن زبان بر روی ماشین‌های مختلف است.
۸. محیط برنامه نویسی: یکی از امکاناتی که جذابیت برنامه نویسی با آن زبان خاص را بیشتر می‌کند، محیط برنامه نویسی آن زبان است.

## دلایل مطالعه زبان‌های برنامه نویسی مختلف

- زبان‌های برنامه نویسی را می‌توان از جهات مختلفی مورد بررسی قرار داد که چند نمونه از این دلایل در زیر به آن اشاره شده است:
۱. افزایش توانایی به منظور ایجاد و توسعه الگوریتم‌های کار: این بدان معناست که به کمک الگوریتم‌های مختلف می‌توان زبان‌های مختلف با روش‌های مختلفی ساخت، مانند استفاده از شیء‌گرایی یا بازگشتی در ایجاد برنامه‌های مؤثر.

۲. ارتقاء روش‌های استفاده از زبان‌های موجود:  
به طور مثال می‌توان از آرایه‌ها، لیست‌ها و رکوردها در جای مناسب استفاده کرد تا بتوان روش‌های مختلفی از یک زبان را ارائه دهیم.
۳. افزایش لغت شناسی در مورد ساختارهای برنامه نویسی:  
مانند استفاده از هم‌روال‌ها برای اینکه بتوانیم دایره لغات خود را در رابطه با ساختارهای زبانهای برنامه نویسی مختلف افزایش دهیم.
۴. انتخاب بهتر زبان برنامه نویسی:  
این به معنای انتخاب بهترین زبان برنامه نویسی برای آن حرفه مد نظر ما است، مانند استفاده از زبان‌های هوش مصنوعی در کاربردهای هوش مصنوعی و.....
۵. یادگیری آسانتر زبان جدید: یک برنامه نویس به دلیل آشنایی با مفاهیم موجود در زبان‌های مختلف می‌تواند خیلی آسانتر از یک فرد مبتدی زبان برنامه نویسی جدیدی را فرا گیرد.
۶. ساده‌تر شدن طراحی زبان: در این مسئله نیز به دلیل آشنایی با مفاهیم مورد نیاز در زبان می‌توان یک زبان را به سادگی طراحی کرد.



خلاصه نویسی

## بخش دوم

### سیر تکاملی زبان های برنامه نویسی

۱. زبان های با مبنای عددی (FORTRAN, ALGOL)
۲. زبان های تجاری (COBOL)
۳. زبان های هوش مصنوعی (LISP, PROLOG, SNOBOL)
۴. زبان های سیستمی (C, Assembly)

دوره	کاربرد	زبان مورد استفاده
1960	تجاری	COBOL
	علمی	FORTRAN, BASIC, ALGOL, APL
	سیستمی	Assembly
	هوش مصنوعی	LISP, SNOBOL
امروزه	تجاری	C++, Java, 4GL
	سیستمی	Java, C, C++, BASIC
	علمی	C, C++, Java
	هوش مصنوعی	LISP, PROLOG
	انتشارات	TEX, Postscript

### نسل های برنامه نویسی

نسل اول: زبان ماشین که در اواخر دهه ۱۹۴۰ بوجود آمد، در این زبان که تنها زبان قابل فهم برای کامپیوتر می باشد از ارقام صفر و یک به

عنوان غلایم اولیه استفاده می‌شود و ارقام صفر و یک در حقیقت الفبای این زبان محسوب می‌شوند و جهت ایجاد کدهایی برای دستورالعمل‌ها بکار می‌روند، به طوری که هر دستورالعمل به صورت رشته‌ای از صفر و یک‌ها نوشته می‌شود.

نسل دوم: زبان اسمبلی که در اوایل دهه ۱۹۵۰ بوجود آمد، این زبان در واقع همان زبان ماشین است، با این تفاوت که جهت ساده نمودن کار برنامه نویسی، کدهای سمبلیکی به نام کد نیمانیک در اوایل سال‌های ۱۹۵۰ بوجود آمد که در آن، از حروف برای کد گذاری کدهای زبان ماشین استفاده شد که این کدها در ریزپردازنده‌های مختلف با یکدیگر تفاوت دارند. این کدها توسط نرم افزارهای به خصوصی بنام اسمبلر به زبان ماشین تبدیل می‌گردند تا قابل درک برای ماشین باشند.

نسل سوم: زبان‌های سطح بالا که در اواخر دهه ۱۹۵۰ بوجود آمدند و اولین زبان این نسل فرتن بود. به این زبان‌ها زبان‌های رویه‌ای یا رویه‌گرا نیز گفته می‌شود زیرا برنامه نویسی باید چگونگی این عملیات را تشریح نماید.

نسل چهارم: اواسط دهه ۷۰ این زبان‌ها بسیار شبیه به زبان‌های طبیعی می‌باشند و برنامه نویسی بوسیله برنامه‌هایی که به این زبان‌ها می‌نویسد به روشی ساده‌تر از زبان‌های دیگر با کامپیوتر ارتباط برقرار می‌نماید، گویی در حال صحبت کردن معمولی با کامپیوتر می‌باشد. به این زبان‌ها زبان‌های غیررویه‌ای نیز گفته می‌شود زیرا برنامه نویسی بدون تشریح چگونگی عملیات، خواسته خود را مطرح می‌کند. یعنی به کامپیوتر می‌گوید چه می‌خواهد، ولی چگونگی انجام عملیات را نمی‌گوید. به عنوان مثال از این نسل می‌توان به زبان ADA اشاره نمود. به این زبان‌ها، زبان‌های فوق بالا نیز گفته می‌شود.

## تقسیم بندی کلی زبان‌های برنامه نویسی

در کل زبان‌های برنامه سازی به دو دسته تقسیم می‌شوند:

زبان‌های سطح پایین: زبان‌هایی هستند در سطح ماشین و به دور از زبان طبیعی و محاوره‌ای انسان. این زبان‌ها وابسته به ماشین و سخت افزار هستند، بطوری که هر ریز پردازنده زبان خاص خود را داراست. کار کردن با این زبان‌ها مشکل است و خطایابی و بررسی برنامه‌ها به سهولت امکان پذیر نیست، ولی به علت نزدیکی به ماشین، برنامه‌های نوشته شده به این زبان‌ها با سرعت بالایی اجرا می‌شوند.

زبان‌های سطح پایین به دو دسته زبان ماشین و اسمبلی تقسیم می‌شوند. زبان‌های سطح بالا: زبان‌هایی هستند نزدیک به زبان طبیعی و محاوره‌ای انسان که در آن‌ها از علائم، حروف و کلمات آشنا بکار رفته در زبان طبیعی استفاده می‌شود. این زبانها احتیاج به ترجمه و تفسیر دارند تا قابل درک برای کامپیوترها شوند که این امر بوسیله نرم افزارهای بخصوصی بنام کامپایلر و مفسر انجام می‌گیرد. زبان‌های سطح بالا وابسته به ماشین، سخت افزار نیستند و با اندکی تغییر در کلیه کامپیوترها قابل اجرا می‌باشند. البته شرط قابل درک بودن این زبان‌ها برای کامپیوترها وجود مفسر یا کامپایلر می‌باشد. کارکردن با این زبان‌ها آسان‌تر است و نسبت به زبان‌های سطح پایین خطایابی و بررسی برنامه‌ها راحت‌تر انجام می‌گیرد. تولید زبان‌های سطح بالا از اواسط دهه ۱۹۵۰ آغاز گردید و متداولترین آن‌ها عبارتند از: FORTRAN, COBOL, PL/1, BASIC, LOGO, PSCAL, C

## دسته بندی زبان‌های برنامه نویسی

زبان‌های ساخت یافته:

زبان‌های ساخت یافته به زبان‌هایی می‌گویند که از ساختار منظم که توسط توابع ایجاد شده‌اند برخوردار هستند. مثلاً برای نوشتن برنامه‌ای که می‌خواهد دو عدد را باهم جمع کند تابعی به نام‌های add بنویسیم که دو عدد را بگیرد و با هم جمع کند و این تابع را در برنامه دیگرمان هم استفاده کنیم. برخی از زبان‌های ساخت یافته در زیر نوشته شده‌اند:

Haskell: یک زبان برنامه نویسی کاملاً ساخت یافته

ML: یک زبان برنامه نویسی استاندارد که نگارش‌های مختلفی دارد.

Erlang: یک زبان ساخت یافته با کاربرد صنعتی.

J&K: دو زبان برنامه نویسی با قابلیت‌های آرایه‌ای قوی.

APL: یک زبان برنامه نویسی ساخت یافته بر پایه آرایه‌ها.

LISP: یک زبان برنامه نویسی ساخت یافته. البته از این زبان به بعد برنامه نویس شی گرا مطرح شد.

C: این زبان به عنوان اولین زبان برنامه نویسی سیستمی مطرح است.

زبان‌هایی مثل PASCAL و BASIC هم از این گروه هستند.

در ۲۰ سال اخیر کمیته برنامه نویسی ساخت یافته بجای ارائه زبان جدید به ارتقای زبان‌هایی مثل LISP و ML پرداختند.

زبان‌های شی گرا: زبان‌های شی گرا زبان‌هایی هستند که بر پایه اشیاء ساخته و نوشته می‌شوند همه چیز در این زبان‌ها شی محسوب می‌شود. و هم عرض با این زبان‌ها نوعی برنامه نویسی به نام OOP بوجود آمده که این زبان‌ها در واقع نسل جدید زبان‌های برنامه نویسی هستند.

Simula: اولین زبان برنامه نویسی OOP که در سال ۱۹۶۰ عرضه شد.

C++: این زبان در حقیقت ترکیبی از برنامه‌نویسی سیستمی و شی‌گرا است.

Perl: یک زبان برنامه‌نویسی تحت UNIX که برای ساخت وب سایت‌های پویا بکار می‌رود.

PHP: زبان برنامه‌نویسی سمت سرور که در سالهای اخیر محبوبیت زیادی کسب کرده است.

Java: زبان برنامه‌نویسی محصول شرکت Sun  
تعداد زبان‌های برنامه‌نویسی واقعاً زیاد است VB و ASP و ... نیز از این جمله‌اند.

زبان‌های متنی: زبان‌های متنی زبان‌هایی هستند که نحوه نگارش آنها به زبان طبیعی نزدیک است. این گونه زبان‌ها معمولاً پیچیدگی‌های انواع دیگر را ندارند و می‌توان گفت ساده‌تر هستند:

TCL: پدر بزرگ زبان‌های متنی

Perl: یک زبان متنی است که قابلیت‌های گرا دارد

JavaScript: محبوب‌ترین زبان برنامه‌نویسی متنی در حال حاضر برای برنامه‌نویسی سمت مشتری در صفحات وب.

Python: زبان برنامه‌نویسی قدرتمند با قابلیت‌های گرایبی بسیار قوی برای برنامه‌نویسی سمت مشتری در صفحات وب.

زبان‌های منطقی: زبان‌های برنامه‌نویسی منطقی به نوعی سردمدار سبک جدیدی از برنامه‌نویسی هستند که در علوم مختلف کاربرد دارد:

PROLOG: اولین زبان برنامه‌نویسی متنی که در سال ۱۹۷۲ ایجاد شد.

Mercury: زبان برنامه‌نویسی منطقی دیگر ...

## مدل‌های برنامه‌نویسی

دستوری: در این زبان‌ها باید بطور دقیق چگونگی تبدیل ورودی‌ها به خروجی‌ها و عملیات‌های لازم بر روی خانه‌های حافظه را مشخص کرد. مانند زبان‌های C و پاسکال

تابعی: توجه به عملی که باید انجام شود، بدون توجه به چگونگی انجام آن عمل. به طور معمول ظاهر زبان‌های تابعی به صورت فراخوانی تعدادی تابع، و ارسال نتیجه آنها به عنوان پارامتر به توابع دیگر است. زبانی مانند C ترکیبی از زبان تابعی و دستوری است.

قانونمند: انجام عملیات با برقراری شروطی در برنامه بدون توجه به ترتیب اجرای آنها. مانند PROLOG.

شیء‌گرا: اولویت داشتن داده‌ها نسبت به عملیات مورد نیاز بر روی آنها. در اینگونه زبان‌ها، اشیاء مورد نیاز، در برنامه ساخته شده به گونه‌ای که عملیات مورد نیاز بر روی آنها در درون هر شیء گنجانده شده است. مانند ++C که ترکیبی از تابعی، دستوری و شیء‌گراست.

## بخش سوم

### مروری بر تاریخچه چند زبان برنامه نویسی

#### معتبر

#### زبان برنامه نویسی C

در آزمایشگاه BELL در اوایل دهه ۱۹۷۰ به منظور تکمیل و باز نویسی نسخه اولیه سیستم عامل UNIX طراحی شد و امروزه نسخه‌های مختلفی از زبان بوجود آمده است. گرچه C یک زبان سطح بالا است ولی غالباً به عنوان زبان برنامه نویسی سیستم و یا برای رفع نیازهایی که در گذشته به کمک زبان اسمبلی برطرف می‌شدند استفاده می‌شود. همچنین بسیاری از نرم افزارهای اساسی کامپیوتر به این زبان نوشته می‌شوند. فراگیری این زبان برای مبتدیان کار دشواری است.

#### زبان برنامه نویسی LOGO

این زبان توسط سیمور پاپرت در دهه ۱۹۶۰ در دانشگاه MIT عرضه شد. گرچه این زبان جهت استفاده دانشجویان به منظور کارهای علمی طراحی گردید، لیکن آن را به عنوان اولین زبان آموزشی جهت پرورش مهارت و خلاقیت بچه‌ها می‌شناسند. رسم خطوط گرافیکی، کار روی رنگ‌ها، ایجاد تصاویر متحرک در این زبان بسادگی انجام می‌شود.



## زبان برنامه‌نویسی LISP

توسط جان مک آرتی در سال ۱۹۵۹-۱۹۶۰ به منظور پشتیبانی تحقیق در زمینه هوش مصنوعی ارائه گردید و روی داده‌های غیر عددی کار می‌نماید و جهت برنامه‌نویسی در محیط AUTO CAD نیز مورد استفاده قرار می‌گیرد.

## زبان برنامه‌نویسی ALGOL

این زبان در سال ۱۹۵۸ معرفی گردید و یک زبان علمی می‌باشد. نسخه‌های مختلفی از ALGOL تا کنون عرضه شده است که از جمله می‌توان ALGOL68 را نام برد. در آمریکا معمولاً از FORTRAN بجای ALGOL استفاده می‌شود، ولی در اروپا این زبان از محبوبیت ویژه‌ای برخوردار است.

## زبان برنامه‌نویسی PASCAL

این زبان که به افتخار بلز پاسکال دانشمند فرانسوی قرن هفدهم میلادی، پاسکال نامگذاری شده است در اواخر سال‌های ۱۹۶۰ و اوایل ۱۹۷۰ توسط پروفیسور نیکلاس ویرث در انستیتو فنی فدرال سوئیس مطرح گردید. این زبان از قدرت بالایی در اجرای امور علمی و تجاری برخوردار است و در بسیاری از مدارس و کالج‌های دنیا جهت آموزش برنامه‌نویسی تدریس می‌گردد و در سال ۱۹۸۳ توسط سازمان استاندارد ملی آمریکا بصورت استاندارد در آمد.

## زبان برنامه نویسی ADA

این زبان به افتخار نام دختر لرد بایرون که همکار چالز بابیج در زمینه طرح ماشین تحلیلی بود، ADA نامیده شد. خانم ADA را بخاطر برنامه هایش به عنوان اولین برنامه نویسی در جهان می‌شناسند. این زبان به منظور سرویس‌های نظامی در وزارت دفاع آمریکا تهیه گردید. در سال ۱۹۷۵ وزارت دفاع آمریکا تحقیقاتی را پیرامون طراحی یک زبان عمومی که مورد استفاده فروشندگان کامپیوتر و برنامه نویسان نظامی باشد آغاز کرد که ماحصل کار آن‌ها زبان ADA بود که در سال ۱۹۸۰ عرضه گردید. این زبان در سال ۱۹۸۳ توسط سازمان ملی استاندارد آمریکا به صورت استاندارد درآمد.

## زبان برنامه نویسی Basic

بیسیک به معنی زبان همه منظوره برای افراد مبتدی می‌باشد. این زبان به خاطر ساختار ساده‌ای که دارد از محبوبیت فوق العاده‌ای در جهان برخوردار است. در سیستم‌های محاوره‌ای و اشتراک زمانی استفاده می‌شود. یک زبان محاوره‌ای امکان ارتباط مستقیم بین انسان و کامپیوتر را در حین اجرای برنامه فراهم می‌نماید. یک فرد مبتدی که آشنایی چندانی با کامپیوتر ندارد پس از مدت کوتاهی می‌تواند دستورهای این زبان را فرا گرفته و اقدام به نوشتن برنامه بنماید. وارد کردن داده‌ای ورودی بسیار ساده بوده و برنامه نویسی لازم نیست نگران دستورهای (فرمت) خروجی برنامه باشد زیرا فرمت‌های خروجی قابل استفاده توسط این زبان در اختیار است. همچنین ایجاد تغییرات و اضافه کردن داخل برنامه بیسیک بسادگی انجام می‌شود. به خاطر سادگی این زبان، BASIC

در اولین میکرو کامپیوترها مورد استفاده قرار گرفت و تا کنون نیز محبوب‌ترین زبان سطح بالای مورد استفاده در این سیستم‌های شخصی برای آموزش نو آموزان می‌باشد. زبان بیسیک بین سال‌های ۱۹۶۳ و ۱۹۶۴ توسط پروفسور جان کمنی و توماس کورتز در کالج دارتموث بوجود آمد و هدف آن‌ها از ایجاد زبان بیسیک این بود که کلیه دانشجویان رشته‌های مختلف بتوانند آن را بسادگی فرا گیرند. علیرغم اینکه در بیسیک اولیه از دستورات معین و محدودی استفاده می‌شد، لیکن سازندگان کامپیوتر دستورات متعددی را به آن افزودند و از نظر سخت افزاری امکانات کامپیوتر خود را افزایش دادند تا بتوانند با سایر سازندگان کامپیوتر رقابت نمایند، لذا امروزه نسخه‌های متعددی از بیسیک وجود دارد و سازمان استاندارد آمریکا نسخه‌ای از آن را بنام نسخه پایه در سال ۱۹۷۸ ارائه نمود. استاندارد فوق به حدی ساده است که نسخه‌های گسترش یافته آن از قبیل QBASIC, GWBASIC, TURBO BASIC, VISUAL BASIC قابل دسترسی است. اینک از دانش آموزان مدارس تا مهندسان هواپیما از این زبان استفاده می‌نمایند. همچنین این زبان در امور تجاری و مدیریت کاربرد دارد.

## زبان برنامه نویسی COBOL

کوبول به معنی زبان تجاری می‌باشد که برای پردازش فایل‌ها بوجود آمد و هم اکنون برای کارهای تجاری با حجم زیاد مورد استفاده قرار می‌گیرد. در سال ۱۹۵۹ بسیاری از نمایندگان دولت آمریکا و سازندگان و استفاده کنندگان کامپیوتر و دانشگاه‌ها گرد هم آمدند تا زبان مناسب برای پردازش فایل‌ها را بوجود آورند. حاصل کار آن‌ها در ژانویه سال ۱۹۶۰ به اتمام رسید. مشخصات این زبان چند ماه بعد توسط سازمان انتشارات دولتی به ثبت رسید و در سال ۱۹۶۱ کامپایلر زبان کوبول برای امور تجاری عرضه شد. سازمان ANSI استاندارد برای زبان کوبول در سال ۱۹۶۸ تهیه کرد و در سال ۱۹۷۴ نسخه جدیدی از آن نیز عرضه شد. زبان کوبول بهتر از سایر زبان‌های برنامه نویسی قادر به انجام عملیات بر روی کاراکترهای الفبایی از قبیل نام، آدرس و سایر مشخصات دیگر می‌باشد و محدودیت آن این است که برای انجام عملیات پیچیده ریاضی مناسب نمی‌باشد.

## زبان برنامه نویسی PL/1

همانطور که ملاحظه نمودید زبان‌های اولیه از قبیل فرترن و کوبول به منظور حل مسائل علمی، تجاری تهیه گردیدند. اما در اوایل دهه ۱۹۶۰ شرکت IBM و یک کمیته از استفاده کنندگان IBM 360 کار خود را بر روی زبانی که قابلیت فرترن و کوبول را توأم داشته باشد آغاز نمودند که PL/1 نامیده شد و در اواسط دهه ۱۹۶۰ کار تهیه این زبان به پایان رسید. PL/1 نیز مانند یک زبان علمی از برخی تکنیک‌های فرترن و کوبول بهره جست و سازمان استاندارد آمریکا (ANSI) در سال ۱۹۷۶

استانداردی برای آن تهیه کرد. علیرغم اینکه PL/1 زبان پر قدرتی می‌باشد و به این منظور ساخته شد که جایگزین فرترن و کوبول گردد لیکن از آنجایی که فراهم نمودن تکنیک‌هایی که قادر به انجام امور علمی و تجاری باشد مشکل است، لذا PL/1 موفقیت مورد نظر را کسب ننمود. با توجه به اینکه فراگیری این زبان نیز ساده نیست می‌رود که به تدریج به دست فراموشی سپرده شود.

## زبان برنامه نویسی FORTRAN

فرترن اولین زبان سطح بالا است که تولید آن در سال ۱۹۵۴ به سرپرستی جان باکوز به منظور ایجاد یک زبان علمی در شرکت IBM شروع و در سال ۱۹۵۷ روی IBM 704 معرفی گردید که بالغ بر ۲/۵ میلیون دلار هزینه برداشت. با استفاده از این زبان حل معادلات ریاضی بسیار آسان گردید و بسیار مورد استقبال قرار گرفت. این زبان در اکثر کامپیوترهای بزرگ و کوچک مورد استفاده قرار می‌گیرد و همین استقبال فوق العاده سبب شد تا کار تهیه استاندارد در سال ۱۹۶۲ برای آن شروع شود که یکی از آن‌ها را نسخه پایه و دیگری را نسخه کامل یا گسترش یافته می‌نامند. استاندارد زبان فرترن در سال ۱۹۶۶ مورد پذیرش سازمان استاندارد آمریکا قرار گرفت و این اولین زبانی بود که به صورت استاندارد درآمد. برنامه‌هایی که به این زبان در یک کامپیوتر نوشته می‌شود معمولاً به سادگی در سایر کامپیوترها نیز قابل استفاده می‌باشد. فرترن نیز از دستورات ورودی، خروجی، محاسباتی، منطقی / مقایسه‌ای و سایر دستورات اساسی از قبیل STOP و READ,WRITE,GOTO همانطور که از این دستورها در زبان انگلیسی انتظار می‌رود استفاده می‌شود. زبان فرترن قابلیت حل مسائل ریاضی و

آماري را دار مي‌باشد، لذا بسياري از برنامه‌هاي اين مقوله به اين زبان نوشته مي‌شوند. از ضعف‌هاي اين زبان اين است كه دنبال كردن منطق برنامه مشكل‌تر از ساير زبان‌هاي سطح بالا مي‌باشد و اين زبان براي پردازش فايل‌ها نمي‌باشد، لذا براي پردازش فايل‌ها و استفاده در امور تجاري زبان سطح بالاي ديگري بوجود آمد به نام كوبول.

خلاصه نویسی

## بخش چهارم

### هوش مصنوعی از چند دیدگاه مختلف

هوش مصنوعی به سیستم‌هایی گفته می‌شود که می‌تواند واکنش‌هایی مشابه رفتارهای هوشمندانه انسانی، از جمله درک شرایط پیچیده، شبیه سازی فرآیندهای تفکری و شیوه‌های استدلالی انسانی و پاسخ موفق به آنها، یادگیری و توانایی کسب دانش و استدلال برای حل مسئله را داشته باشد. همچنین به هوشی که یک ماشین در شرایط مختلف از خود نشان می‌دهد، نیز گفته می‌شود. بیشتر نوشته‌ها و مقالات مربوط به هوش مصنوعی، آنرا به عنوان "دانش شناخت و طراحی عامل‌های هوشمند" تعریف کرده‌اند. یک عامل هوشمند، سیستمی است که با شناخت محیط اطراف خود، شانس موفقیت خود را پس از تحلیل و بررسی افزایش می‌دهد.

جان ماکرتی که واژه هوش مصنوعی را در سال ۱۹۵۶ استفاده نمود، آنرا "دانش و مهندسی ساخت ماشین‌های هوشمند" معرفی کرده است. تحقیقات و جستجوهای انجام شده برای رسیدن به ساخت چنین ماشین‌هایی با بسیاری از رشته‌های علمی در ارتباط و همکاری است؛ مانند علوم رایانه، روان‌شناسی، فلسفه، عصب‌شناسی، علوم ادراکی، تئوری کنترل، احتمالات، بهینه‌سازی و منطق می‌باشد.



در پژوهش‌کننده شرکت IBM نیز تعریفی با این مضمون ارائه شده است که می‌گوید: "هوش مصنوعی بعنوان زیر شاخه‌ای از کامپیوتر محسوب شده و ارتباطی تنگاتنگی با عصب شناسی، علوم شناختی، روانشناسی شناختی، منطق ریاضی و مهندسی دارد."

هربرت سیمون (دارای جایزه نوبل اقتصاد ۱۹۷۸ و جایزه تورینگ ۱۹۷۵) نیز می‌گوید: "هوش مصنوعی عبارت است از ایجاد ظرفیت برای انجام وظایفی که عموماً بعنوان ویژگی‌های انسان شناخته می‌شود. در کامپیوتر این ظرفیت‌ها شامل: استدلال، اکتشاف، مفهوم، تعمیم، یادگیری و... می‌باشد."

بطور کل می‌توان هوش مصنوعی را اینگونه معنی کرد که؛ هوش مصنوعی روشی است در جهت هوشمند کردن کامپیوتر تا قادر باشد در هر لحظه تصمیم‌گیری کرده و اقدام به بررسی یک مسئله نماید. هوش مصنوعی، کامپیوتر را قادر به تفکر می‌کند و روش آموختن انسان را تقلید می‌نماید. بنابر این اقدام به جذب اطلاعات جدید جهت بکارگیری مراحل بعدی می‌پردازد. مغز انسان به بخش‌هایی تقسیم شده که هر بخش وظیفه خاص خود را جدا از بقیه انجام می‌دهد. اختلال در کار یک بخش تأثیری در دیگر قسمت‌های مغز ندارد. در برنامه‌های هوش مصنوعی نیز این مسئله رنایت شده است در حالی که در برنامه‌های غیر هوش مصنوعی مانند C و Pascal تغییر در برنامه روی سایر قسمت‌های برنامه و اطلاعات تأثیر دارد.

## مزیت کامپیوترهای هوشمند نسبت به انسان

- خسته نمی‌شود
- گرانی نیروی انسانی
- عدم شک و اشتباه
- انسا نها سر حال نیستند و ...
- عدم فراموشی
- سرعت بالا، دقت بالا و ...

## مهم ترین قابلیت‌های ربات هوشمند

ادراک: این ربات یا Agent باید توانائی درک محیط اطراف خود را داشته باشد (تصویر و صوت). یکی از مسائل مهم دیگر برای ربات‌ها، شناخت از وجود خود (Sentient) بودن است. این بدان معنا است که ربات از وجود خود آگاه بوده و می‌تواند بروی محیط خود اثرگذار باشد. این یکی از ویژگی‌های مهم هر انسان است ولی مدرکی دال بر Sentient بودن حیوانات وجود ندارد تاکنون ربات بسیار کامل که قادر به درک خود باشد به وجود نیامده است. دانشمندان آلمان رباتی به نام Gaak ساخته‌اند که این ربات قادر به فکر کردن به طور مستقل بوده است. این ربات در مسابقات " زنده ماندن قوی‌ترها " سعی نموده از یکی از مبارزاتش بگریزد. این ربات از محل مبارزه گریخته و در تصادفی از بین رفته است. این حادثه می‌تواند دلیلی بر هوشمند بودن این ربات باشد ولی دلیل قاطعی برای اثبات این موضوع نیست.

یادگیری: ربات باید قابلیت یادگیری الگوهای جدید پیرامون خود را داشته باشد. ربات باید بتواند مسائل جدید را بیاموزد و در جایی همانند مغز انسان اطلاعات را نگهداری نماید.

تطابق و پذیرش: با محیط، خود را تطبیق و در صورت تغییر محیط می‌بایست قابلیت پذیرش الگوهای جدید را داشته باشد.

قابلیت استدلال: کامل بودن پایگاه دانش برای استدلال ربات پاسخ دهی مناسب به رویدادها (تصمیم‌گیری درست)

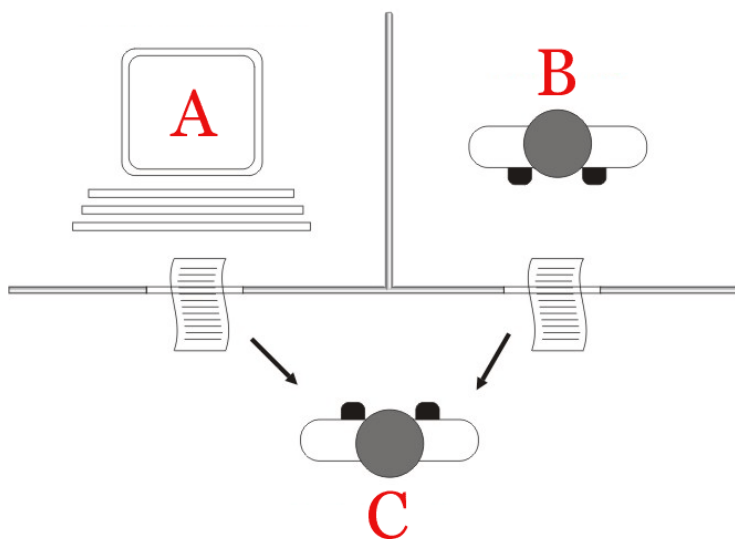
## تاریخچه هوش مصنوعی

هوش مصنوعی به خودی خود علمی است کاملاً جوان. در واقع بسیاری، شروع هوش مصنوعی را در سال ۱۹۵۰ می‌دانند، زمانی که آلن تورینگ مقاله دوران ساز خود را در باب چگونگی ساخت ماشین هوشمند نوشت (آنچه بعدها به تست تورینگ مشهور شد). تورینگ در آن مقاله یک روش را برای تشخیص هوشمندی پیشنهاد می‌کرد، که این روش بیشتر شبیه به یک بازی بود.

## تست تورینگ

فرض کنید شما در یک سمت دیوار (پرده یا هر مانع دیگر) هستید و به صورت تله تایپ با آن سوی دیوار ارتباط دارید و شخصی از آن سوی دیوار از این طریق با شما در تماس است. طبیعتاً یک مکالمه میان شما و شخص آن سوی دیوار می‌تواند صورت گیرد. حال اگر پس از پایان این مکالمه، به شما گفته شود که آن سوی دیوار نه یک شخص بلکه یک ماشین بوده است - شما از آنطرف دیوار هیچ اطلاعاتی ندارید - که

پاسخ شما را می‌داده، آن ماشین یک ماشین هوشمند خواهد بود، در غیر اینصورت، یعنی شما در میان پاسخ‌های آن سوی دیوار به مصنوعی بودن پاسخ‌ها پی ببرید، دیگر به ماشین آن سوی دیوار هوشمند نمی‌گویند. باید دقت کرد که این تست به دو دلیل بسیار مهم انتخاب شد - البته استفاده از متن به جای صوت هم تأثیر گذار است - یک اینکه موضوع ادراکی صوت را کاملاً از صورت مسئله حذف کند و این تست هوشمندی را درگیر مباحث مربوط به دریافت و پردازش صوت نکند و دوم اینکه هوش مصنوعی به سمت نوعی از پردازش زبان طبیعی تاکید کند. تا کنون تلاش‌های زیادی برای پیاده سازی تست تورینگ انجام شده است و برنامه‌هایی مانند Eliza یا AIML نیز برای چت کردن اتوماتیک ساخته شده است، اما هیچ یک از ماشین‌ها تا کنون قادر به گذر از این تست نبوده است.



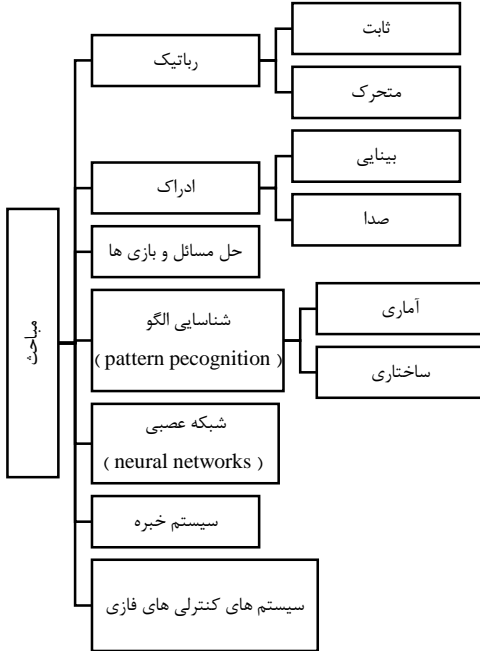
## عوامل موفقیت در تست تورینگ

برای موفقیت در تست تورینگ کامپیوتر باید قابلیت‌های زیر را داشته باشد:

- پردازش زبان طبیعی: درک معنایی جملات نوشته شده برای محاوره به زبان انگلیسی. (مشکل است مثلاً ضرب المثل‌ها را نمی‌فهمد...)
- بازنمایی دانش: اطلاعات تولید شده قبل یا در حین آزمون را ذخیره سازد.
- استدلال خودکار: از اطلاعات ذخیره شده برای پاسخ به پرسش‌ها استفاده کرده و نتایج جدیدی را استخراج نماید.
- یادگیری ماشین: خود را با شرایط جدید وفق داده و الگوها را کشف کند.
- پیش فرض‌های اساسی تست تورینگ عبارت‌اند از:
  - نمونه کامل هوشمندی انسان
  - توانایی پردازش و درک زبان طبیعی که مهمترین مشخصه هوشمندی است.

## مباحثی هوش مصنوعی

مباحثی که در ارتباط با هوش مصنوعی مطرح می‌شوند:



### سیستم خبره

به آن نوع از برنامه‌های هوش مصنوعی که به سطحی از خبرگی می‌رسند که می‌توانند به جای یک متخصص در یک زمینه خاص تصمیم‌گیری کنند گویند. این سیستم برنامه‌هایی هستند که پایگاه دانش آنها انباشته از اطلاعاتی که انسان هنگام تصمیم‌گیری درباره یک موضوع خاص، براساس آن‌ها تصمیم می‌گیرند. روی این موضوع باید تأکید کرد که هیچ یک از سیستم‌های خبره‌ای که تا کنون طراحی و برنامه نویسی شده‌اند،

همه منظوره نبوده‌اند و تنها در یک زمینه محدود قادر به شبیه سازی فرآیند تصمیم گیری انسان هستند.

### **محدوده وظایف**

به محدوده اطلاعاتی‌ای از الگوهای خبرگی انسان که به یک سیستم خبره منتقل می‌شود گفته می‌شود. این محدوده، سطح خبرگی یک سیستم خبره را مشخص می‌کند و نشان می‌دهد که آن سیستم خبره برای چه کارهایی طراحی شده است. سیستم خبره با این وظایف می‌تواند کارهایی چون برنامه ریزی، زمانبندی، و طراحی را در یک حیطه تعریف شده، انجام می‌دهد.

### **مهندسی دانش**

روند ساخت یک سیستم خبره را مهندسی دانش گویند. یک مهندس دانش باید اطمینان حاصل کند که سیستم خبره طراحی شده، تمام دانش مورد نیاز برای حل یک مسئله را داشته باشد؛ که در غیر اینصورت تمام تصمیم‌های سیستم خبره غیر قابل اطمینان است.

## بخش پنجم

# زبان برنامه نویسی لیسپ

### تاریخچه زبان لیسپ

زبان lisp در اواخر دهه ۵۰ توسط جان مکاریتی<sup>۱</sup> در MIT به وجود آمد. بنابراین زبان از قدیمی ترین زبانهای برنامه سازی (سطح بالا) است. هدف عمده طراحی آن انجام محاسبات نمادین<sup>۲</sup> بود. چون بیشتر محاسبات کامپیوتری در آن زمان به صورت عددی بود و این برای برخی از شاخه های علوم کامپیوتر (مانند هوش مصنوعی) کافی نبود. باید اعتراف کنیم که زبان lisp زبان بسیار جالبی است. یعنی کسی که با lisp و زبانهای نظیر آن آشنایی ندارد، در این زبان ویژگی هایی را خواهد یافت که در زبانهایی که تا کنون با آنها آشنا شده، چنین ویژگی هایی را ندیده است. یکی از بارزترین این ویژگیها یکنواختی فوق العاده ای است که در آن به چشم می خورد و موجب می شود این زبان بسیار انعطاف پذیر باشد. با مطالعه مطالبی که در ادامه خواهد آمد به این مطلب پی می برید. هر زبان برنامه سازی از ترکیب تعدادی واحد اولیه تشکیل شده. مثل زبانی مانند C از عبارات (جمع، ضرب، انتساب و...)، دستورات کنترلی مثل شرطها و

---

<sup>۱</sup> John McCarthy

<sup>۲</sup> symbolic computation



حلقه‌ها، بلک‌ها، توابع و ... تشکیل شده. همچنین دارای تعدادی داده<sup>۱</sup> است. مثل اعداد صحیح، اعداد اعشاری، کاراکترها، رشته‌ها، structure ها و ... ترکیب تمام این اجزا یک برنامه C را به وجود می‌آورند. زبان برنامه نویسی LISP: زبان برنامه نویسی لیسپ یکی از قدیمی‌ترین زبان‌های برنامه نویسی هوش مصنوعی است که می‌خواهیم با آن تا حدودی آشنا شویم. این زبان برنامه نویسی دارای خصوصیات خاصی است که در ادامه به آن‌ها می‌پردازیم.

### S – expression

تمام اجزای سازنده این زبان، چه دستورات و چه داده‌ها همه از نوع S-expression هستند. با توجه به این تعریف می‌توان دریافت که یک داده در لیسپ خود به تنهایی می‌تواند یک برنامه لیسپ باشد. یکی از بارزترین تفاوت‌های این زبان با زبان C این است که در C می‌توان یک متغیر از نوع program تعریف کرد، اما خود این متغیر نیز، متغیر یک برنامه C است. پس از آن نیز می‌توان چند دستور if یا for هم اضافه کرد، بعد با یک تابع آنرا اجرا کرد که همه این کارها در زمان اجرا اتفاق می‌افتد.

s-expression ها در کل به دو دسته اصلی تقسیم می‌شوند که طبق جدول زیر به بررسی آن می‌پردازیم:

<sup>۱</sup> Data

s-expression	
atom (اتم)	list (لیست)
هر اتم می‌تواند رشته‌ای از کارکترها باشد	اعضایش می‌تواند هم یک اتم باشد و هم یک لیست دیگر (مجموعه‌ای از اتم‌ها)
مانند: ۰-۹ A-Z a-z %\$@#%().+ "<>	مانند: ۳ ۲ ۱ I am student
	یک لیست خالی که هم می‌تواند یک اتم باشد و هم یک لیست

برنامه‌های محاسباتی در lisp به صورت prefix نوشته می‌شوند. با توجه به ریاضیاتی که ما تا کنون خوانده‌ایم عملوندها و عملگرها همراه با هم نوشته می‌شوند. در روش prefix از روش زیر استفاده می‌شود:

عملوند عملوند عملگر

۲ ۳ + (مثال)

جواب این عبارت با حالت infix آن برابر است:

$(+ ۳ ۲) = (۳ + ۲)$

نمونه‌ای از atom و list:

**Atom:**

ABC

+

123

Nil ()

Hello-world

**List:**

(1 2 3)

(I am boy)

(A b c (1 2) @)

( ) nil

Lisp زبانی است که عبارات را محاسبه می‌کند، این کار زبان lisp با دیگر زبان‌ها تفاوت دارد، زبان‌های دیگر کنترلی روی روند برنامه دارند و این کنترل را به کمک شرط، حلقه، تعریف توابع و... انجام می‌دهند اما زبان lisp هیچ یک از اسن کارها را انجام نمی‌دهند، زبان lisp فقط تمام این کارها را در قالب محاسبه انجام می‌دهد. زبان C را بدون داشتن حلقه یا شرط تصور کنید، آیا این زبان یکنواخت نبود؟ خوب زبان LISP هم این چنین است، یک زبان یکنواخت (اما از دید من بسیار زیباست). در زبان C عمل انتساب یک عبارت است، نه یک دستور، ولی فرض کنید ساختار If، While، for، switch هم یک عبارت می‌بود. تفاوت زبان C با LISP هم دقیقاً در همین است؛ همه چیز در زبان LISP یک عبارت است، نه یک دستور.

وقتی یک s-expression به عنوان یک برنامه به LISP داده می‌شود تا اجرایش کند، LISP فقط آنرا محاسبه می‌کند، مانند مثال روبرو زیرا LISP فقط می‌تواند محاسبه کند و خبری از دستور نیست، در این مثال نمی‌داند با Print و P چه کند، زیرا هیچ یک برایش تعریف نشده است، تنها راهی که برای حل این مسئله وجود دارد چیزی است به نام مقدار یابی.

مثال:

```
>>> (Print P)
ERROR
```

## مقداریابی در LISP

مقداریابی		
atom (اتم)	list (لیست)	
اگر عدد باشد، همان عدد را بر می‌گرداند	اگر symbol باشد، حاصل مقداری است که به آن نسبت داده اید	اولین عضو را به عنوان تابع می‌گیرد و سعی می‌کند روی بقیه اعضا اعمال کند
مانند: <pre>&gt;&gt;&gt; 1.2 1.2</pre>	مانند: اگر $a = 4$ <pre>&gt;&gt;&gt; (+ 1 2 3 a) 10</pre>	مانند: <pre>&gt;&gt;&gt; (1 2 3 4) ERROR</pre> <p>زیرا در این برنامه عدد ۱ را به عنوان تابع انتخاب می‌کند و روی بقیه اعداد (آرگومان‌ها) اعمال می‌کند، اما ۱ که کاری نمی‌کند زیرا تعریف شده نیست پس ERROR می‌دهد.</p>

## Interpreter

برنامه Lisp باید مانند دیگر برنامه‌ها مقداریابی و اجرا شود، به این منظور سازندگان LISP برنامه‌ای را با عنوان interpreter ساخته‌اند تا بتواند آرگومان‌های برنامه را پس از مقداریابی، در صورت درستی بروی دیگر اعضا اعمال کند.

یک چیز دیگر که در برنامه LISP مهم است، این است که مانند دیگر زبان‌ها توابع و عملگرها دو چیز متفاوت هستند.

چند مثال از مقدار یابی و اعمال (به حالت prefix بودن توجه کنید):

```
>>> (+ 1 2)
```

```
3
```

```
>>> (* (+ 7 3) (- 7 3))
```

```
40
```

حال که با بعضی از اصطلاحات و کارایی‌های زبان LISP آشنا شدید، می‌خواهم شما را با چند تابع هم آشنا کنم که کارایی‌ها و جذابیت‌های خاصی را به برنامه اضافه می‌کند. در ادامه این مطالب نیز با تابع نویسی و ساخت توابع جدید آشنا می‌شوید.

### بخشی از اجرای برنامه LISP

```
lisp 4.34 Copyright (c) 1997-2013 Ufasoft
[1]> (* (+ 7 3) (- 7 3))
40
[2]> |
```

### تابع LIST

تابع LIST مجموعه‌ای از آرگومان‌ها را می‌گیرد و عیناً آنها را بر می‌گرداند. شاید پیش خود بگویید که این تابع چه فایده‌ای دارد، اما با آشنایی بیشتر با این برنامه می‌توانید کاربرد این تابع را حس کنید.

```
>>> List (1 2 3 4)
```

(1 2 3 4)

**تابع LENGTH**

این تابع تعداد آرگومان‌های ما را شمارش می‌کند. حال تصور می‌کنیم که ما این تابع را می‌شناسیم، اما با تابع LIST آشنایی نداریم، در اینصورت این اتفاق به این دلیل رخ داد که interpreter فکر می‌کند که ۱ را باید روی دیگر آرگومان‌ها اعمال کند، و چون ۱ تعریف نشده است پس ERROR می‌دهد. اما برای درست شدن این مشکل باید اینگونه نوشت:

```
>>> Length (1 2 3 4)
ERROR
```

حال می‌بینید که تعداد شمارش شده توسط تابع length درست بوده است زیرا interpreter ابتدا به کمک تابع list اعداد (آرگومان‌ها) را باز می‌گرداند و سپس با تابع length آن‌ها را شمارش می‌کند!

```
>>> Length (List (1 2 3 4))
4
```

**single quote یا (')**

برای اینکه در LISP بگوییم قبل از اعمال تابع روی آرگومان‌های خود، آن‌ها را مقدار یابی کن می‌توانیم از (') که یک single quote است استفاده کنیم، کاری که این عبارت انجام می‌دهد، دقیقاً همان کاری است که تابع List انجام می‌دهد. پس:

```
>>> Length (List (1 2 3 4))
4
```

پس

```
>>> Length '(1 2 3 4)
4
```

مثال: در اینجا با تابع Print آشنا می‌شوید که وظیفه‌اش چاپ آن آرگومان خاص است.

```
>>> (Print 'salaam)
Salaam
```

مثال:

```
>>> (+ 1 2)
3
```

پس

```
>>> '(+ 1 2)
(+ 1 2)
```

شاید تا کنون خودتان متوجه این مطلب شده باشید که زبان LISP اهمیت فراوانی برای List های خود (نه تابع list) دارد، شاید با کمی موشکافی بتوان متوجه این مطلب شد که این زبان از کلمات: LISP = LIST Processing تشکیل شده است که این مطلب نشان می‌دهد چقدر پردازش list در LISP مهم است.

## تابع (CAR (FIRST / HEAD)

تابع car یک آرگومان از list گرفته و اولین عضو آنرا برمی‌گرداند.

مثال:

```
>>> (car '(+ 1 2))
+
>>> (car '(1 2 a 3))
1
>>> (car ())
ERROR
```

```

lisp 4.34 Copyright (c) 1997-2013 Ufasoft
[1]> (car '(1 2 3 4))
1
[2]> (car '(+ 2 3))
+
[3]> (car '((1 2) 2 (3)))
(1 2)
[4]> (car '())
NIL
[5]>

```

## بخشی از اجرای برنامه LISP

### تابع CDR (REST / TAIL)

این تابع یک آرگومان از نوع list گرفته و یک List شامل اعضای دوم به بعد را برمی‌گرداند.  
مثال:

```

>>> (cdr '(+ 1 2))
(1 2)
>>> (cdr '(a 1 2 c 5))
(1 2 c 5)
>>> (cdr ())
ERROR

```

### تابع CONS

یک symbol یا یک s-expression و یک list می‌گیرد، و یک list برمی‌گرداند که عضو اول آن (s-expression) symbol است و عضو دوم آن اعضای list است. حال می‌خواهیم یک مثال ترکیبی را با هم پاسخ دهیم:

```

>>> (cons 1 '(2 3 4))
(1 2 3 4)

```



```
>>> (car (cdr '(1 2 3)))
```

```
2
```

و اما اتفاقی که در interpreter می افتد اینگونه است:

```
>>>cdr '(1 2 3)
```

```
(2 3)
```

```
>>>car '(2 3)
```

```
2
```

خوب فراموش که نکردید، بحث ما در رابطه با IA یا همان هوش مصنوعی است! حال می‌خواهیم با lisp برنامه‌ای برای یک ربات بنویسیم و ببینیم پاسخ او چیست؟!

مثال: اگر پیگاه داده ربات ما این آرگومان‌ها باشد که به ترتیب (نام خانوادگی، نام، شماره دانشجویی) پس خواهیم داشت:

```
((800111 Ali Mohseni)
```

```
(800112 Reza Nazari)
```

```
(800113 Maryam Karimi))
```

حال اگر سؤال من از ماشین این باشد که: نام خانوادگی نفر دوم لیست چیست؟

جواب ماشین باید این چنین باشد:

```
>>> (car (cdr (cdr (car (cdr L))))))
```

```
Nazari
```

و اما اتفاقی که در interpreter می افتد اینگونه است:

```
Abcd Efgh Ijk
```

```
Cdr '(A E H) = E H
```

```
Car '(E H) = E
```

```
Cdr '(f g h) = g h
```

```
Car '(g h) = g
```

```
b = 800111 / c = Ali / d = Mohseni: A
```

```
e = 800112 / f = Reza / g = Nazari: E
```

```
h = 800113 / I = Maryam / j = karimi: I
```

یکی دیگر از قابلیت‌های LISP این است که شما می‌توانید Macro هم داشته باشید، این بدین معناست که می‌توان مثال ربات بالا را به این صورت نوشت تا از هدر روفتن وقت برای تایپ جلوگیری کنید:

```
>>> (car (cdr (cdr (car (cdr L))))))
```

```
Nazari
```

پس

```
>>> (caddadr L)
```

```
Nazari
```

پس از یادگرفت این مطالب و تمرین با این المان‌ها باید به سراغ مطالب سخت‌تر برویم. `nil` یا `()` در LISP نشان دهنده یک مقدار منطقی نادرست است. خود `nil` نیز یک لیست خالی را نمایش می‌دهد، پس در نتیجه تفاوتی ندارد که چگونه نوشته شود:

```
>>> ' nil
```

```
Nil
```

پس

```
>>> nil
```

```
nil
```

در LISP مقدار صحیح با `T` نمایش داده می‌شود. در این زبان هر عبارتی بجز `nil` یک مقدار منطقی درست به حساب می‌آید. از آنجایی که مقدار منتسب به `nil` همان `nil` است، همین رابطه نیز برای `T` هم صدق می‌کند.

```
>>> T
```

```
T
```

## تابع ATOM

یک آرگومان گرفته و مقدار منطقی آنرا بر می‌گرداند و نشان می‌دهد که آرگومان یک اتم atom است (T) یا نه (nil) و به یاد داشته باشید nil به تنهایی یک اتم است.  
مانند:

```
>>> atom 'x
T
>>> atom '(1 2 3)
Nil
>>> atom nil
T
```

در LISP توابعی که مقدار بازگشتی آنها مقداری منطقی باشد/ نباشد، Predicate می‌گویند.

## تابع EQ

این تابع دو آرگومان گرفته و می‌گوید آیا دو آرگومان مربوط به یک نقطه از حافظه هست یا نه! دو symbole مساوی که در یک آدرس از حافظه قرار دارند، یعنی در جایی به نام atom table قرار دارند. این مکان در زبان‌های دیگر با نام Symbol table نام برده شده است. در صورتی یک symbol به atom table اضافه می‌شود که در آنجا نباشد. در برخی از موارد دو list با هم مساوی هستند، اما آدرسی برابر ندارند که با این تابع می‌توان این مطلب را دریافت.

```
>>> eq 'X 'X
T
باشد  $X = Y$  در صورتی که
>>> eq 'X 'Y
```

T

&gt;&gt;&gt; eq '(1 2) '(1 2)

nil

محاسبه در زبان LISP اینگونه تعریف می‌شود: احضار تابع و اجرای هر تابع روی آرگومان‌های خودش. تعریف این جمله را می‌توان در مثال روبرو معنی کرد.

&gt;&gt;&gt; LIST (1 2 3)

(1 2 3)

### تابع عملیات شرطی COND

آرگومان‌های این تابع شامل تعدادی list دو عضوی به شکل زیر است:

یک عبارت یک عضو منطقی

Cond (condition1 exp1) (condition2 exp2).....

این تابع در حقیقت، exp عضوی که منطقی باشد (T باشد) را باز می‌گرداند.... این تعریف را بهتر است با یک مثال بیشتر توضیح دهیم:

&gt;&gt;&gt; COND ((eq x 'a) 'aa)

((eq x 'b) 'bb)

((eq x 'c) 'cc)

(T 'nn))

cc

دو نکته در این مسئله وجود دارد:

- x به c انتساب داده شده یعنی x و c دارای یک آدرس در حافظه هستند.
- خط آخر این برنامه - (T 'nn) - نیز یکی دیگر از قوانین عملیات شرطی در زبان LISP است.

تا اینجا با برخی از توابع و کارایی‌های آن‌ها آشنا شدید، اما هنوز یادنگرفتید چگونه می‌توانیم یک تابع تولید کنیم، در ادامه با انواع ساخت توابع آشنا می‌شویم و مثال‌هایی از توابعی که می‌توانیم بسازیم نیز

آموزش داده می‌شود. یاد گرفتن هیچ زبانی آسان نیست و نیاز به تمرین فراوان دارد، زیرا با تمرین می‌توانیج تجربه کسب کنید و با داشتن تجربه حتماً می‌توانید برنامه‌ای بسازید. در زیر جدولی از توابعی که تا کنون آموخته‌اید را می‌بینید:

کاربرد	نام تابع
برگرداندن آرگومان‌های یک لیست یا اتم	LIST - ' مثال:
مثال: >>>LIST (1 2 3 4) (1 2 3 4)	مثال: >>>'X X
شمارش تعداد اعضا	LENGTH
مثال: >>>length (1 2 3 a) 4	
خشی کننده LIST	Eval (مخالف LIST)
مثال: >>>eval '(+ 1 2) 3	
چاپ	Print
مثال: >>>print 'salam Salam	
نگه داشتن آرگومان اول و حذف بقیه	CAR
مثال: >>>car '(1 2 3) 1	
حذف عضو اول و نگه داشتن بقیه	CDR
مثال: >>>cdr '(1 2 3) (2 3)	
یک symbol یا یک s-expression و یک list می‌گیرد، و یک list برمی‌گرداند که عضو اول آن symbol (s-expression) است و عضو دوم آن اعضای list است	CONS
مثال: >>>cons 1 '(a b 3) 1 a b 3	

کاربرد	نام تابع
یک آرگومان گرفته و مقدار منطقی آنرا بر می‌گرداند و نشان می‌دهد که آرگومان یک اتم (atom) است (T) یا نه (nil)	ATOM
مثال: <pre>&gt;&gt;&gt; atom '(1 2) Nil</pre>	مثال: <pre>&gt;&gt;&gt; atom 'a T</pre>
این تابع دو آرگومان گرفته و می‌گوید آیا دو آرگومان مربوط به یک نقطه از حافظه هست یا نه	EQ
مثال: <pre>&gt;&gt;&gt; eq 'x 'x T</pre>	اگر $T = EQ 'x 'c$ باشد
تابع عملیات شرطی	COND
مثال: <pre>&gt;&gt;&gt; cond ((eq x 'c) 'cc) (T 'nn) ) Cc</pre>	

## ساخت تابع به روش define و defun

- تعریف lambda: چون در زبان برنامه‌نویسی LISP حلقه وجود ندارد به همین دلیل از روش‌های مختلفی برای جبران آن استفاده می‌کنیم. حال لامبدا به روش زیر ساخته می‌شود که بعدها از آن در ساخت تابع به روش define از آن استفاده می‌کنیم:

```
>>> Lambda (x y) (+ x y) 2 3
```

## تعریف تابع به روش define

برای مثال ما می‌خواهیم برنامه‌ای به نویسیم که دو تابع به نام‌های mns و pls که اولی برای جمع آرگومان‌ها و دومی برای تفاضل آن‌ها باشد،

چنین برنامه‌ای را به روش زیر می‌نویسیم، یادتان باشد که این یک نمونه است، اما قالب نوشتن توابع دیگر همینگونه است.

مثال: نام تابعی که می‌خواهیم بسازیم pls, mns

```
>>> (define '
(pls (lambda (a b) (+ a b))
(mns (lambda (a b) (- a b))))
```

باتوجه به تابعی که ساختیم خواهیم داشت:

```
>>> (pls 2 5)
7
```

یا

```
>>> (mns (pls 3 6) (mns 6 9))
12
```

## تعریف تابع به روش defun

خوبی این روش به این است که می‌توانیم از دستوری مثل if هم - در قالب عبارت - در این حالت استفاده کرد. برای درک بهتر با مثال به دست آوردن n! (فاکتوریل) که هم در ساختمان داده و هم در طراحی الگوریتم کاربرد دارد کار می‌کنیم:

```
>>> defun fact (n) نام تابع (n)
```

```
(if (< n 2) شرط تابع (< n 2)
```

```
1
```

```
(* n (fact (- n 1)))
```

## بخش هشتم

### زبان برنامه نویسی پرولوگ

برای نوشتن انواع برنامه روش‌های مختلفی وجود دارد یکی از این روش‌ها روش توصیفی است که در آن برنامه نویس فقط بر روی توصیف یا تعریف منطق حل مسئله تاکید می‌کند و ترتیب انجام عملیات برای رسیدن به هدف، بر عهده زبان می‌باشد. زبان‌هایی نظیر LDL و PROLOG و LISP از این نوع زبان‌ها هستند. چنانچه زبان برنامه نویسی بر روی خصوصیات حل مسئله بیشتر تاکید داشته باشد، زبان منطقی نامیده می‌شود، مانند LDL و PROLOG.

#### ویژگی‌های زبان توصیفی

- زبان‌های توصیفی زبان‌هایی هستند که چگونگی انجام مراحل برای رسیدن به پاسخ برای برنامه نویس غیرقابل دسترس بوده و این کار برعهده زبان است. این ویژگی باعث می‌گردد اینگونه زبان‌ها برای نوشتن برنامه‌هایی که منطق پیچیده‌ای دارند، مناسب‌تر باشد.
- در زبان‌های توصیفی، برنامه نویسی عملیاتی نظیر مدیریت حافظه، تعریف ساختمان داده‌های موردنیاز، استفاده از اشاره گرها را لازم نیست انجام بدهیم و این عملیات برعهده زبان است.



- به کارگیری منطق در برنامه‌ها باعث می‌شود که بتوانیم داده‌ها را در برنامه‌ها به شکل یک fact یا بدیهیات بیان کنیم و یا به صورت یکسری قواعد یا rules بیان کنیم.
- در برنامه نویسی توصیفی کافی است برنامه نویس بتواند صورت مسئله را توصیف کند.

### حل مسائل در هوش مصنوعی

شیء خاص: اگر یک ساختاری داشته باشیم که بتوانیم رفتارش را مشخص کنیم، یعنی رفتارش خاص باشد، به آن ساختار شیء می‌گویند. ما معمولاً مسائل را به روش دانش<sup>۱</sup> حل می‌کنیم و در رابطه با مسائل هوشمند نیز همین دانش به کمک ما می‌آید. دانش در دوچیز خلاصه می‌شود، یک حقایق<sup>۲</sup> و یکی قوانین حاکم بر حقایق<sup>۳</sup>؛ و بغیر از همه این المان‌ها، تجربه نیز بسیار پر کاربرد است. حل با دانش و تجربه‌ای که از یک مسئله داریم ما را به هدف مسئله رسانده و باعث حل آن می‌شود. مسئله اصلی هوش این است که کدام مسیر را برای حل، انتخاب کنیم - هوش یعنی استنتاج و قضاوت با استفاده از دانش و تجربه در مورد یک مسئله - که در نهایت برای مسائل، از سیستم تولید شده استفاده می‌کنیم؛ این سیستم دارای ۳ مرحله اساسی است:

۱. پایگاه داده (fact ها)

۲. قوانین تولید (rul ها)

<sup>۱</sup> Knowledge

<sup>۲</sup> facts

<sup>۳</sup> rules

## ۳. استراتژی کنترل (انتخاب بهترین مسیر)

## ویژگی‌های زبان پرولوگ

هر برنامه در پرولوگ فقط از تعدادی گزاره خبری (fact) و تعدادی گزاره شرطی (rule) تشکیل شده است. Rule ها fact ها را در در محدوده مسئله به یکدیگر مرتبط می‌سازد و این گزاره‌ها در زمان اجرا توسط موتور زبان ارزیابی می‌گردد. قلمرو دسترسی به متغیرها در این زبان بسیار ساده می‌باشد. در PROLOG نیازی به تعریف متغیرها نیست و هر متغیر فقط در محدوده‌ی همان گزاره‌ای که در آن به کار رفته است قابل دسترسی می‌باشد؛ این خصوصیات باعث می‌گردد که احتمال بروز خطا کاهش یابد. برنامه پرولوگ - ۵ تا ۱۰ برابر - کوچک‌تر از برنامه‌های معادل خود در زبان‌های رویه‌ای (منطقی) می‌باشد. این خصوصیات باعث می‌گردد که حجم برنامه کم شود و احتمال بروز خطا کاهش یابد، با توجه به این مزایا نیز هزینه تغییر و نگهداری برنامه کاهش می‌یابد.

این زبان فقط برای کاربرد هوش مصنوعی نیست، بلکه خود به تنهایی یک زبان همه منظوره قوی به شمر می‌آید که بر روی انواع محیط‌ها قابل پیاده سازی می‌باشد. زبان پرولوگ برای استفاده در کاربردهای پردازش زبان‌های طبیعی<sup>۱</sup> نیز استفاده می‌شود. که این پردازش زبان طبیعی خود به دو مرحله تقسیم می‌شود:

- دریافت صدا و تغییر صدا به حروف و کلمات
- درک معنایی جملات یا کلمات نوشته شده به زبان طبیعی

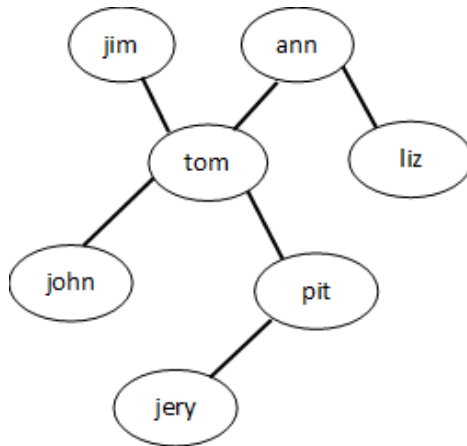
<sup>۱</sup> N.L.P: (Natural Language Processing)

## گزاره‌های خبری<sup>۱</sup> و گزاره‌های شرطی<sup>۲</sup>

هر برنامه هوش مصنوعی دارای تعدادی Fact است که همان صورت مسئله یا داده‌های ما از آن مسئله هستند، اما چون با یک زبان منطقی / توصیفی کار می‌کنیم باید قوائدی برای مسئله طرح کنیم که در صورت درستی آنرا حل کند که Ruleها همان قوائد هستند.

مسائل معروف برای آشنایی با Rule و Fact: معروف‌ترین مسئله‌ای که می‌توان برای آشنایی افراد با گزاره‌ها در زبان PROLOG یافت، مثال تعریف یک خانواده و آشنایی با روابط فامیلی بین آنها است که در ادامه بطور کامل با آنها آشنا می‌شوید ...

Fact های مسئله: ابتدا روابط فامیلی افراد را اینگونه به تصویر می‌کشیم، سپس سؤالات و درخواست‌هایی در مورد رابطه خانوادگی بین این افراد را از PROLOG می‌پرسیم:



<sup>۱</sup> Fact

<sup>۲</sup> Rule

نکته: در پرولوگ اسامی خاص با حرف کوچک نوشته می‌شوند. رابطه فامیلی: ما به طور قرارداد رابطه‌ای به شکل  $\text{parent}(X, Y)$  را در نظر می‌گیریم که در آن همیشه ( $X$  والد  $Y$  است) یا  $Y$  فرزند  $X$  است) که با توجه به قراردادی که ما در بالا توضیح دادیم، رابطه بین افراد به صورت ۶ گزاره بیان می‌شود:

Parent (jim, tom).

Parent (ann, tom).

Parent (ann, liz).

Parent (tom, john).

Parent (tom, pit).

Parent (pit, jery).

نکته: برای اجرای برنامه بالا می‌توانید کدهای گفته شده را در یک note pad کپی کنید و سپس آنرا با فرمت (.pl) ذخیره کنید.

اگر به خاطر داشته باشید در کودکی بازی با ما می‌کردند که در آن از ما سؤال‌هایی در رابطه با روابط خانوادگی می‌شد، به طور مثال "دایی شما چه نسبتی با شما دارد؟" و پاسخ ما اینچنین بود: "برادر مادرم" می‌شود دایی ما "ماشین نیز مانند یک کودک است و باید برایش این روابط را خیلی ساده و کودکانه توضیح داد.

## نوشتن درخواست در پرولوگ

اگر بخواهیم در رابطه با گزاره‌های بالا سؤالی بپرسیم می‌توانیم از روش‌های زیر استفاده کنیم:

مثال:

فرزند ann کیست؟

? - parent (ann, x).

X = liz X = tom

آیا tom پدر Liz است؟

? - parent (tom, liz).

No

نوه‌های jim کیست؟

? - parent (jim, X), parent (X, Y).

X = tom Y = john

X = tom Y = pit

## اضافه کردن روابط دیگر (قاعده جنسیت)

اگر بخواهیم در مورد روابط دیگر خانوادگی بین افراد سؤالاتی طرح کنیم که به صورت fact در برنامه نوشته می‌شوند. برای این منظور ابتدا جنسیت افراد را در برنامه به صورت یکسری fact مشخص می‌کنیم.

## قوانین جنسیت

در پرولوگ به دو روش می‌توان قوانین جنسیت را یادداشت نمود:

Male (X) - female (X)

Sex (X, male) - sex (X, female)

پس از اینکه جنسیت افراد را در برنامه مشخص کردیم، می‌توانیم به راحتی روابط زیر را توضیح دهیم، روابط زیر یکسری Rule هستند.

## دستور شرطی (-): در پرولوگ

مثال‌های زیر نمونه‌هایی از دستورات شرطی در پرولوگ هستند که برای تعریف روابط خانوادگی استفاده می‌شوند:

خواهر tom کیست؟

? - sister (A,tom).

A= liz

رابطه خواهر بودن به این شکل مشخص می‌شود:

Sister (X,Y):- female (X), parent(Z,X), arent(Z,Y).

رابطه پدر بودن به این شکل مشخص می‌شود:

Father (X,Y):- parent(X,Y), male(X).

رابطه مادر بودن به این شکل مشخص می‌شود:

Mother (X,Y):- parent(X,Y), female(X).

رابطه خاله / عمه بودن به این شکل مشخص می‌شود:

ابتدا باید معنی خاله (خواهر مادر) را تعریف کنیم.

Sister (X,Y):- female(X), parent(Z,X), parent(Z,Y).

Aunt (X,Y):- female(X), parent(Z,Y), sister(X,Z).

رابطه دایی بودن به این شکل مشخص می‌شود:

Uncle(X,Y):- parent (Z,Y), male (X), female (Z), parent (A,X),  
parent (A,Z).

### قاعده بازگشتی

اگر سمت راست قاعده از همان رابطه سمت چپ قاعده استفاده کند، به

آن قاعده بازگشتی گویند. برای درک بهتر این قاعده را با یک رابطه

توضیح می‌دهیم:

رابطه جد به این شکل مشخص می‌شود:

Jad (X,Y):- parent(X,Y).

و یا اینکه

Jad (X,Y):- parent(Z,Y), jad(X,Z).

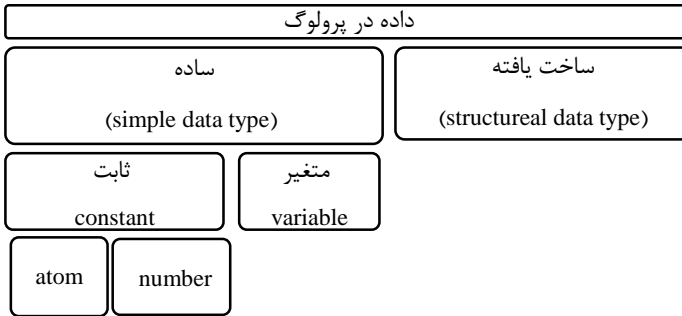
مثال: اجداد jery را بیابید؟

?- jad (X,jery).

X = pit X = tom X = jim X = ann

## انواع داده در پرولوگ

اگر یادتان باشد توضیح دادیم که داده‌ها در زبان لیسپ به دو دسته‌اتم و لیست تقسیم می‌شوند که در رابطه با کاربردها و انواع نوشتاری آنها آشنا شدیم؛ اکنون در زبان پرولوگ هم می‌خواهیم به بررسی انواع داده‌ها بپردازیم و با توجه به شناخت آنها بتوانیم با آنها کار کنیم. در کل داده‌ها در این زبان به دو دسته کلی تقسیم می‌شوند:



### داده ثابت در پرولوگ

Atom: رشته‌ای از حروف، ارقام و علائم که با یک حرف کوچک شروع می‌شود. اگر نخواهیم از این قانون پیروی کنیم می‌توانیم آنرا داخل یک ' ' نوشت، مانند:

al#

ali

'Max'

'ali Reza'

Number: برای نمایش ثابت‌های عددی به کار می‌رود. این داده برای اینکه اعداد را دقیقتر حساب کند، اعداد اعشاری را گرد می‌کند، مانند:

$$1.2 = 1$$

1.9 = 2

## داده متغیر در پرولوگ

متغیرها نقاطی از حافظه هستند که در زمان اجرای برنامه مقداردهی می‌شود - یا مقدار آنها تغییر می‌کند؛ در زبان PROLOG اسامی متغیرها و نوع آنها تعریف نمی‌شود. متغیرها در این زبان اینگونه تعریف می‌شوند که هر رشته از ارقام و حروف و کارکترها که با یک حرف بزرگ شروع شوند، درغیراینصورت باید قبل آنها یک ( \_ ) بیاید. اگر یک متغیر فقط یکبار دریک گزاره استفاده شود به آن متغیر ناشناس گویند و آنرا با (\_\_\_) نشان می‌دهند، مانند:

A  
G#2  
Max  
\_ali

به طور مثال:

Father(X):- male(X), parent(X, \_\_\_).

که در اینجا علامت \_\_\_ به معنی (هرکسی) است.

## داده ساخت یافته

یک داده مرکب است که از انواع داده‌های مختلف تشکیل شده است و توسط یک عملگر<sup>۱</sup>-این اجزا داده‌ای- به یکدیگر نسبت داده می‌شوند. روش نوشتن داده به شکل روبرو است:

Functor (اجزاء ساختار).

Functor: رشته‌ای از حروف و ارقام و علائم است که با یک حرف کوچک شروع می‌شود.

<sup>۱</sup> Functor



اجزاء ساختار: می‌تواند هریک از انواع داده باشد؛ حتی خود داده ساخت یافته

data ((1393, mehr, Day), Ann, 80009856, arya).

انواع داده:

داده ساخت یافته = (1393, mehr, Day)

داده متغیر = Ann

داده number = 80009856

داده atom = arya

### قواعد کلی تطبیق

عمل تطبیق<sup>۱</sup>: مهم‌ترین عملی که زبان پرولوگ بر روی داده‌های یک برنامه انجام می‌دهد، عمل تطبیق است.

<p>T و S</p> <p>دو گزاره هستند که می‌خواهیم آنرا با هم تطبیق دهیم</p>		
<p>T و S</p> <p>دو داده ثابت باشند</p>	<p>S یک متغیر</p> <p>و</p> <p>T هر چه می‌خواهد</p>	<p>T و S</p> <p>دو داده ساخت یافته باشند</p>
<p>در صورتی با هم match می‌شوند که دقیقاً یکسان باشند</p>	<p>T هر چه که باشد مهم نیست و با S تطبیق می‌شود، ابتدا متغیر مقدار T مقداردهی می‌شود و بلعکس</p>	<p>۱. عملگرشان یکسان باشد</p> <p>۲. اجزاء و مؤلفه‌های آن نظیر به نظیر بایکدیگر تطابق داشته باشند</p>
<p>S = 12a</p> <p>=</p> <p>T = 12a</p>	<p>S = 12A</p> <p>=</p> <p>T = 5</p>	<p>S = (12, a, A)</p> <p>=</p> <p>T = (B, f, 8)</p>

<sup>۱</sup> matching

داده‌های برنامه:

data1 (1391, mehr, Day).

data2 (1391, 7, 23).

data3 (year, Mon, 23).

سؤال این است که کدام مؤلفه‌ها با هم تطابق دارند؟

جواب اینگونه است که، data1 با data2 تطابق ندارد اما با data3 تطابق دارد. به این دلیل چنین جوابی می‌گیریم چون data1 و data2 دارای دو نوع مختلف از داده‌ها هستند و اجزاء آنها نظیر به نظیر با هم تطابق ندارد، اما این تطابق در data3 وجود دارد زیرا داده‌های data3 نیز از انواع (Number و متغیر و atom) می‌باشد. روش تطبیق نیز روش سوم است (تطبیق از نوع ساخت یافته).

## معنای توصیفی و رویه‌ای برنامه‌ها در پرولوگ

فرض کنید سه عبارت به نام‌های P, Q, R, دارید و بخواهیم گزاره‌ای به شکل P:- Q, R را بنویسید، دو روش برای خواندن این برنامه وجود دارد:

- توصیفی: P صحیح است به شرطی که Q و R هم صحیح باشند.
  - رویه‌ای: برای بدست آوردن P باید Q و R را بدست آورد.
- در روش رویه‌ای (منطقی) ترتیب انجام عملیات برای رسیدن به پاسخ مهم است و معنای توصیفی برنامه مشخص می‌کند که درخواست داده شده صحیح است یا خیر و یا اینکه به ازای چه مقادیری از متغیرها هدف داده شده درست است.

علامت‌های بین گزاره‌ای (،) و (؛)

علامت (،) بین گزاره‌ها به معنای AND است.

علامت (؛) بین گزاره‌ها به معنای OR است.

علامت (;) بین گزاره‌های بدنه<sup>۱</sup> یک عبارت، آنرا به چند عبارت مختلف تبدیل می‌کند:

P:- Q; R.

P:- Q.

P:- R.

اگر عبارتی هم علامت (,) و هم (;) را داشته باشد، آنگاه تقدم علامت (,) بالاتر از (;) است.

P:- Q, R; S, T.

P:- Q, R.

P:- S, T.

مثال:

رابطه خانوادگی X و Y که: یا X والد Y باشد. یا X فرزند Y باشد. یا X و Y داری یک والد مشترک باشند. یا X و Y یک فرزند مشترک داشته باشند.

Relative (X,Y):- parent(X,Y).

Relative (X,Y):- parent(Y,X).

Relative (X,Y):- parent(z,X), parent(z,Y).

Relative (X,Y):- parent(X,w), parent(Y,w).

حال با توجه به علائمی که خوانده‌اید:

Relative (X,Y): parent(X,Y); parent(Y,X); parent(Z,X),  
parent(Z,Y); parent(X,W), parent(Y,W).

همانطور که گفته شد معنای رویه‌ای برنامه مشخص می‌کند PROLOG چگونه به درخواست‌ها پایان دهد و یا چگونه سعی در پاسخ به مجموعه اهداف می‌نماید. هنگامی مجموعه اهداف یک درخواست به دست می‌آورد که متغیرهایی که در مجموعه اهداف قرار دارند بتوانند با یک متغیر ثابت جایگزین شوند.

---

<sup>۱</sup> body

## مجموعه‌ای از تمرین‌های توصیفی و رویه‌ای

در ادامه انواع مثال‌ها را می‌بینید که به دو روش رویه‌ای و توصیفی حل شده‌اند و در پایان هر پاسخ نیز به بررسی جواب رویه‌ای آن می‌پردازیم. مثال (FACT) شماره ۱ در برنامه پرولوگ را با هم تمرین می‌کنیم:

روش نگارش نوع حیوان + اندازه:

Big(bear). / Big(elephant). / Small(cat).

روش نگارش رنگ حیوان:

Brown(bear). / Black(cat). / Gray(elephant).

روش نگارش سؤال اینکه رنگ dark چیست؟

Dark(X):- black(X). / Dark(X):- brown(X).

مثال: کدام حیوان (سیاه / قهوه) ای و بزرگ است؟

?- dark(x), big(x).

در این جا ما به دو روش به مسئله پاسخ می‌دهیم، ابتدا به روش توصیفی و سپس به روش رویه‌ای که به شرح زیر است:

پاسخ توصیفی:

X = bear

```

File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 7.1.10)
Copyright (c) 1990-2014 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
% c:/Users/ASH/Desktop/New Text Document (2).pl compiled 0.00 sec, 9 clauses
1 ?- dark(X), big(X).
X = bear.

2 ?-

```

بخشی از اجرای برنامه Prolog

پاسخ رویه‌ای: در این روش مرحله به مرحله طبق جدول زیر به پاسخ‌های احتمالی ماشین می‌پردازیم تا ببینیم چطور به نتیجه مطلوب خواهد رسید.

شرط اول	شرط دوم	بررسی مسئله
Big(x)	Dark(x)	-
-	Black(x)	ماشین یکبار dark را black فرض می‌کنیم و در صورت درست بودن سراغ بقیه می‌رویم
-	Black(cat)	تنها حیوان black در این سؤال cat است اما black big cat نداریم
-	-	در این مرحله ماشین از رابطه خارج شده و مجدداً مسئله را از ابتدا به روش دیگری حل می‌کند
Big(x)	Dark(x)	-
-	Brown(x)	ماشین یکبار dark را brown فرض می‌کنیم و در صورت درست بودن سراغ بقیه می‌رویم در اینجا ما تنها حیوان brown مان bear است
Big(bear)	-	-
Big(bear)	Brown(bear)	حال ماشین به دنبال حیوانی است که big در موردش صدق کند و هم brown پس پاسخ می‌دهد جواب = bear است

در نهایت ماشین به این نتیجه می‌رسد که:

$$X = \text{bear}$$

مثال (FACT) شماره ۲ در برنامه پرولوگ را با هم تمرین می‌کنیم؛ لطفاً

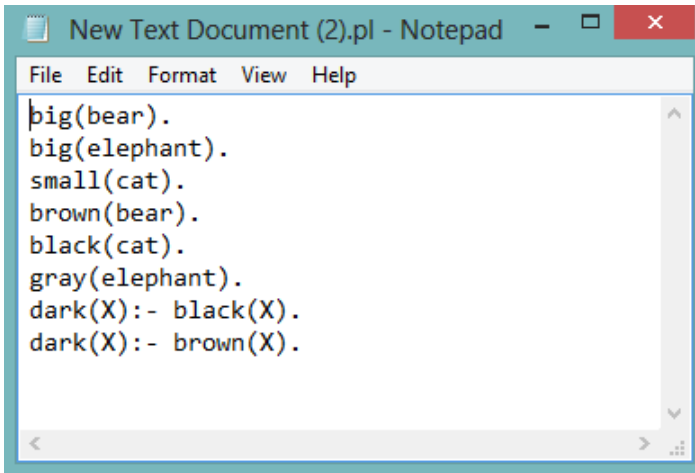
با توجه به روابط بالا به سؤال زیر پاسخ دهید:

مثال: کدام حیوان بزرگ و (سیاه / قهوه) ای است؟

$$?- \text{big}(x), \text{dark}(x).$$

پاسخ توصیفی:

$$X = \text{bear} \quad X = \dots\dots$$



```

File Edit Format View Help
big(bear).
big(elephant).
small(cat).
brown(bear).
black(cat).
gray(elephant).
dark(X):- black(X).
dark(X):- brown(X).

```

### بخشی از اجرای برنامه Prolog

پاسخ رویه‌ای: در این روش مرحله به مرحله طبق جدول زیر به پاسخ‌های احتمالی ماشین می‌پردازیم تا ببینیم چطور به نتیجه مطلوب خواهد رسید.

شرط اول	شرط دوم	بررسی مسئله
Dark(x)	Big(x)	-
Black(bear)	-	-
Black(bear)	big(bear)	تنها حیوان big black در این سؤال bear است اما black big bear نداریم
-	-	در این مرحله ماشین از رابطه خارج شده و مجدداً مسئله را از ابتدا به روش دیگری حل می‌کند
	big(bear)	-
Brown(bear)	-	-
Brown(bear)	Big(bear)	حال ماشین به دنبال حیوانی است که big در موردش صدق کند و هم brown

پس پاسخ می‌دهد جواب = bear است

در نهایت ماشین به این نتیجه می‌رسد که:

$X = bear$

اما ماشین کارش را به همین جا ختم نمی‌شود و به بررسی ادامه می‌دهد تا مطمئن شود پاسخ درست دیگری پیدا نمی‌شود، پس له شکل جدول زیر پاسخ‌های ماشین را مورد بررسی قرار می‌دهیم:

شرط اول	شرط دوم	بررسی مسئله
-	Big(elephant)	-
Black(elephant)	-	-
Black(elephant)	Big(elephant)	تنها حیوان big black در این سؤال elephant است اما black big elephant نداریم
-	-	در این مرحله ماشین از رابطه خارج شده و مجدداً مسئله را از ابتدا به روش دیگری حل می‌کند
-	big(elephant)	-
Brown(elephant)	-	-
Brown(elephant)	Big(elephant)	تنها حیوان big black در این سؤال elephant است اما black big elephant نداریم
-	-	در این مرحله ماشین از رابطه خارج شده و مجدداً مسئله را از ابتدا به روش دیگری حل می‌کند
		اما دیگر چون هیچ حیوان با شرط big نداریم ماشین از رابطه خارج می‌شود و همان آخرین جواب را نمایش می‌دهد

## انواع لیست در پرولوگ

یک لیست مجموعه‌ای از انواع داده‌های مختلفی است که به ترتیب خاصی در لیست قرار می‌گیرند. نحوه نوشتن لیست به شکل زیر است:

[..... جزء دوم, جزء اول]

### سرلیست (HEAD) و دنباله لیست (TAIL)

هر لیست برای تعریف به دو بخش تقسیم می‌شود. اگر درس ساختمان داده را به یاد بیاورید متوجه می‌شوید که زبان پرولوگ نیز از لیست‌های پیوندی<sup>۱</sup> استفاده می‌کند.

- Head: سمت چپ‌ترین عضو لیست است. (سرلیست یک لیست می‌تواند بیش از یک عضو باشد)
- Tail: لیستی است متشکل از بقیه عناصر لیست بجز عضو اول (head) آن.

به طور مثال اگر داده‌های ما به شکل زیر باشند:

$L1 = [1, 2, 3]$

$L2 = [\text{ali}, \text{Family}, \text{data}(13, \text{ali}, \text{nima})]$

$L3 = [[a], b, c]$

برای درک صحیحی از لیست‌ها توجه شما را به مثال‌های زیر جلب می‌کنم:

Head  $L1 = 1$

tail  $L1 = 2, 3$

Head  $L2 = \text{ali}$

tail  $L2 = \text{Family}, \text{data}(13, \text{ali}, \text{nima})$

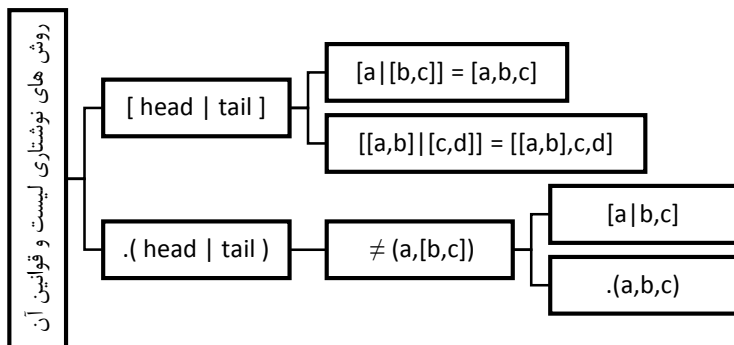
Head  $L3 = [a]$

tail  $L3 = b, c$

<sup>۱</sup> Linked list



## انواع روش‌های لیست نویسی



چنانچه بخواهیم با توجه به روابطی که در بالا ذکر شد مثالی بزینم می‌توانیم به رابطه زیر اشاره کنیم که بدین شکل است:

$$.(a,.(b,.(c,[]))) = [a, b, c]$$

## رابطه Concat

شکل رابطه	Concat(l1,l2,l3)
عملکرد	لیست l3 از لیست l2 به دنبال l1 بدست می‌آید
ضابطه‌های مورد نیاز	اگر لیست اول تهی باشد، لیست سوم برابر لیست دوم است: Concat([],l,l)
	اگر لیست اول تهی نباشد، لیستی خواهیم داشت به شکل [H,T] آنگاه H لیست سوم برابر H لیست اول است و T لیست سوم از الحاق لیست دوم به T لیست اول بدست می‌آید: Concat([H T],l1, [H T2]):_ concat(T1,l1,T2)

مثال: چه مواردی باید باهم الحاق شوند تا داشته باشیم [a,b,c]؟

?-concat(X,Y, [a,b,c]).

x=[]	y=[a,b,c]
x=[a]	y=[b,c]
x=[a,b]	y=[c]
x=[a,b,c]	y=[]

## رابطه sublist

شکل رابطه

Sublist(I1,I2)

عملکرد	اگر I2 زیرلیستی از I1 باشد = true در غیر این صورت false
--------	---------------------------------------------------------

اگر برای sublist های مطرح شده در زیر داشته باشیم:

?-sublist([a,b,c,d], [b,c]).

True

?-sublist([a,b,c,d], [c,d]).

False

پس در نتیجه برای سؤال اصلی زیر خواهیم داشت:

?-sublist([a,b,c],X).

L3 = []	X = []	L5 = [a,b,c]
L3 = []	X = [a]	L5 = [b,c]
L3 = []	X = [a,b]	L5 = [c]
L3 = []	X = [a,b,c]	L5 = []
L3 = [a]	X = []	L5 = [b,c]
L3 = [a]	X = [b]	L5 = [c]
L3 = [a]	X = [b,c]	L5 = []
L3 = [a,b]	X = []	L5 = [c]
L3 = [a,b]	X = [c]	L5 = []

## رابطه subset

شکل رابطه

Subset(I1,I2)

عملکرد	اگر I2 زیر مجموعه‌ای از I1 باشد = True در غیر این صورت False
--------	--------------------------------------------------------------

اگر لیست اول تهی باشد، زیر مجموعه‌های آن نیز تهی می‌شود:

Subset([], [])

ضابطه‌های

اگر لیست تهی نباشد و دارای n عضو باشد، به شکل [H|T] یکبار

مورد نیاز

زیر مجموعه‌های n-1 عضوی T را بدست می‌آوریم:

Subset([H|T], T2):- subset(T,T2).

و یکبار با زیر مجموعه‌های لیست T را به همراه H می‌نویسیم:

Subset([H|T1], [H|T2]):- subset(T1,T2)

مثال:

?-subset([1,2,3],X).

X = []
X = [3]
X = [2]
X = [2,3]
X = [1,3]
X = [1,2]
X = [1,2,3]

در زبان PROLOG جابجایی اهمیت دارد. در مجموعه‌ها، جابجایی عضوهای مجموعه، تاثیری در مجموعه ندارد ولی در لیست‌ها جابجایی اعضا باعث تغییر لیست می‌شود.

## اپراتورها و عبارات محاسباتی در زبان

### عملگر محاسباتی

IS (مساوی)	+
?_ x is 3+2	-
x = 5	
?_ 3+2 = 5	*
False	/

### اپراتورهای محاسباتی و منطقی

;	AND	= / =	:= =	>
OR		(مخالف بودن)	(تساوی)	<
				>=
				=<

## رابطه sumlist

شکل رابطه

Sumlist(l,x)

مجموع مقادیر عناصر لیست l را بدست آورده و در x قرار دهد

Sumlist([], []).

ضوابط

Sumlist([H/T],x):\_ sumlist(T,y), x is H+y

مثال: مجموع عناصر یک لیست تهی برابر صفر است، مجموع یک لیست  $n$  عضوی به شکل  $H$  و  $T$  برابر است با مجموع عناصر لیست  $n-1$  عضوی.

?-sumlist ([3,7,5],X).

H = 3 T = [7,5]

?-Sumlist ([7,5,X]).

H = 7 T = [5]

?-sumlist ([5],X).

H = 5 T = []

## Backtracking در زبان پرولوگ

یکی از خصوصیات خوب زبان PROLOG این است که عمل <sup>۱</sup>backtracking را به صورت اتوماتیک انجام می‌دهد. منظور از backtracking این است که هنگامی که در یک گزاره از برنامه رابطه‌ای استفاده می‌شود، پس از بررسی آن برنامه حتی اگر به جواب رسیده باشد پس از بررسی آن رابطه به محل همان گزاره‌ای که رابطه در آن استفاده شده است باز می‌گردد و اجرای برنامه از همان نقطه ادامه می‌یابد.

### عملگر cut (!)

عملگر cut برای جلوگیری از backtracking است، عملگر cut باعث می‌شود که در صورت رسیدن به جواب در رابطه فراخوانی شده، از بازگشت به عقب جلوگیری گردد و اجرای برنامه قطع گردد.

مثال:

Facts: P(1). / P(2). / P(3).

<sup>۱</sup>بازگشت به عقب در زمان اجرای برنامه

بدون عملگر cut:

 $?-p(X).$  $X=1 \quad X=2 \quad X=3$ 

مثال:

Facts:  $P(1). / P(2). =! / P(3).$ 

با عملگر cut:

 $?-p(X).$  $X=1 \quad X=2$ 

## مثال‌های کاربردی در زبان لیسپ و پرولوگ

نتایج یکسری مسابقات به صورت  $\text{win}(A,B)$  در دست می‌باشد. به این معنا که بازیکن A از B برده است. می‌خواهیم بازیکنان را به سه گروه تقسیم کنیم:

۱. گروه winner یا برندگان که کل مسابقات خود را برده باشند.

۲. گروه player که تعدادی بازی را برده و تعدادی را باخته باشند.

۳. گروه loser یا بازنده که کل بازی‌های خود را باخته‌اند.

حال اطلاعات بالا به صورت فرمول زیر در خواهد آمد:

```
Class (x, player):_ win(x, _), win (_, x),!
```

```
Class (x, winner):_ win (x, _),!
```

```
Class (x, loser).
```

## بررسی n! در دو زبان LISP و PROLOG

در ساختمان داده درسی با عنوان توابع بازگشتی خواندیم، اکنون می‌خواهیم فرمول تابع بازگشتی  $n!$  را یاد بگیریم، سپس با دو زبان آنرا حل می‌کنیم.  
اگر  $n=1$  یا  $n=0$ :

```
Fact(n) = 1
```

اگر  $n > 1$  :

$\text{Fact}(n) = n * \text{fact}(n - 1)$

مثال شماره ۱: اگر نام تابع  $\text{fact}(n)$  باشد برای رابطه  $n!$  خواهیم داشت:

$n!$

if ( $n < 2$ )

return 1

else

return ( $n * \text{fact}(n-1)$ )

پس  $n!$  در LISP به این شکل نوشته می‌شود:

```
>>> (defun fact(n)
```

```
(if (< n 2)
```

```
1
```

```
(* n (fact (- n 1))))
```

```
)
```

مثال شماره ۲: اگر رابطه  $\text{fact}(N,X)$  بطوری که فاکتوریل عدد صحیح  $n$

را بدست آورید و در  $X$  قرار دهید. ضوابط آن به این شکل مطرح می‌شود:

۱. فاکتور ۱ برابر است با ۱.

۲. فاکتوریل  $n$ ، برابر است با فاکتوریل  $n-1$  که پس از محاسبه در  $n$

ضرب می‌کنیم.

پس  $n!$  در prolog به این شکل نوشته می‌شود:

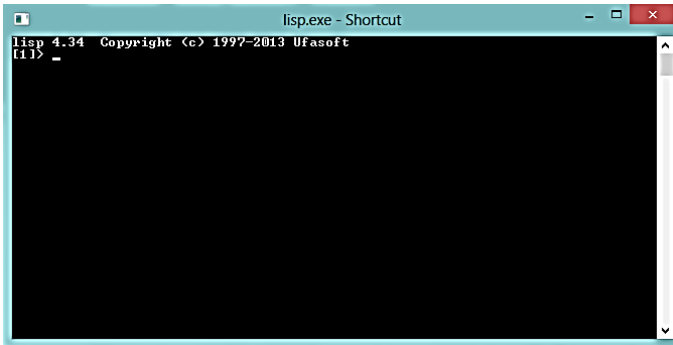
$\text{fact}(1,1).$

$\text{fact}(N,X):- \text{fact}(M,Y), M \text{ is } N-1.$

$?-\text{fact}(X,Y).$

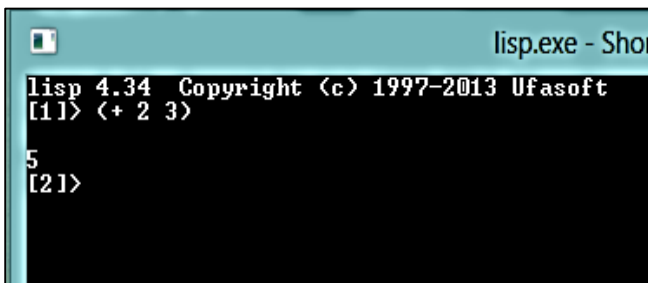
## چگونه زبان لیسپ را اجرا کنیم؟

اجرای برنامه‌های LISP به این شکل است که ابتدا باید برنامه‌ای مانند Common LISP را دانلود کنید، سپس برنامه‌ای به شکل زیر باز می‌شود:



```
lisp.exe - Shortcut
lisp 4.34 Copyright (c) 1997-2013 Ufasoft
[11] > _
```

در این زبان بزرگی یا کوچکی حروف و یا حتی space هم تفاوت ندارد! در محیط باز شده می‌توانیم برنامه‌ای مانند (3 + 2) را به صورت زیر می‌نویسیم تا جواب را به ما بدهد.



```
lisp.exe - Sho
lisp 4.34 Copyright (c) 1997-2013 Ufasoft
[11] > (+ 2 3)
5
[21] >
```

تعریف توابع در LISP (این برنامه از روش define پیروی نمی‌کند و باید توابع را به صورت defun تعریف کرد!) تعیین یک تابع برای جمع دو عدد به نام sum و از روش defun:

```

lisp 4.34 Copyright (c) 1997-2013 Ufasoft
[1]> (defun sum (x y) (+ x y))

SUM
[2]> (sum 1 5)

6
[3]>

```

به نوع نوشتن توجه کنید:

مثال: روش lambda در define است که در ابتدا نام تابع، سپس تعداد عباراتی که باید بکند، و کاری که باید انجام دهد.

```
>(defun sum (x y) (+ x y))
```

```
Sum
```

در روش مقداردهی یادآور شدیم که در list اولین کارکتر به عنوان تابع بر روی دیگر اجزا اعمال می‌شود.

```
>(sum 1 5)
```

```
6
```

## چگونه زبان پرولوگ را اجرا کنیم؟

اجرای برنامه‌های PROLOG به این شکل است که در ابتدا باید برنامه SWI-PROLOG را دانلود کنید سپس با توجه به اطلاعاتی که در جزوه داده شده و با (\*) نمایش داده شده است آنها را اجرا کرد. در همین ابتدا باید متذکر شوم که این برنامه دارای دو محیط است، یک محیط که باید FACT ها را در آن تعریف کرد که در برنامه notepad که در تمام سیستم عامل‌های windows موجود است را در آن نوشت، سپس باید از قسمت file / new ... برنامه‌ای را که با notepad را نوشته‌اید را باز کنید



{ یادتان باشد که باید فرمت Notepad را به (pl.) تغییر دهید، برای این کار کافی است روی نوت پد راست کلیک کرده و گزینه rename را بزیند سپس با ماوس روی (text.) را هایلایت کرده (نگه داشتن چپ کلیک و کشیدن روی قسمت مورد نظر) سپس آنرا به (pl.) تغییر دهید { اگر برنامه مشکلی نداشته باشد (مشکلات به صورت نارنجی در می‌آیند) صفحه باز شده را کوچک می‌کنیم و دوباره از محل اصلی برنامه از قسمت file / consult ... برنامه نوت پد را باز می‌کنیم، اگر درست باشد، یک پیغام سبز می‌نویسد، در صورت درستی شما می‌توانید پرسش خود را در قسمت (?\_) تایپ کنید. در زبان پرولوگ بزرگی یا کوچکی حروف و یا حتی space هم تفاوت دارد!

```

File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 7.1.10)
Copyright (c) 1990-2014 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- █

```

## بخشی از اجرای برنامه Prolog

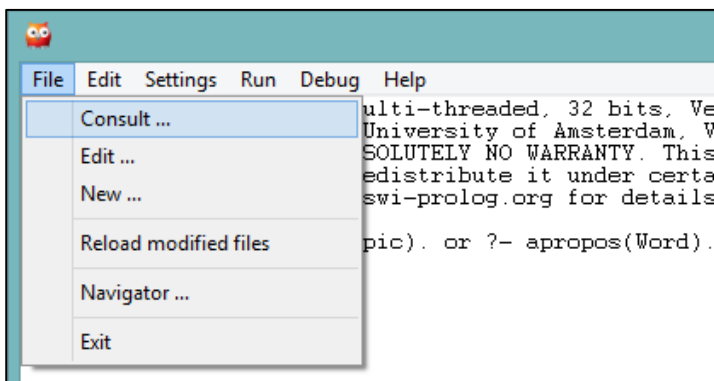
```

File Edit Settings Run Debug Help
multi-threaded, 32 bits, Version 7.1.10
University of Amsterdam, VU Amsterdam
ABSOLUTELY NO WARRANTY. This is free so
edistribute it under certain conditio
swi-prolog.org for details.

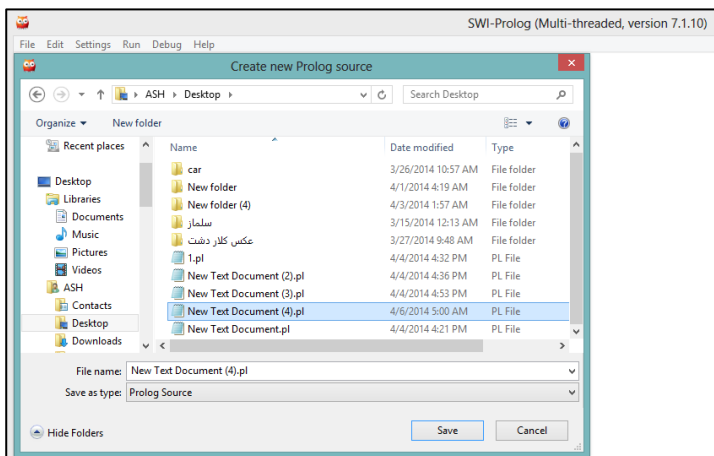
pic). or ?- apropos(Word).

```

مرحله اول اجرای برنامه: آماده سازی برای اجرای یک پروژه جدید



مرحله دوم اجرای برنامه: انتخاب کدهای برنامه



مرحله سوم: پیدا کردن notepad ای که کدهای پرولوگ را در آن  
یادداشت نموده‌اید

اگر برنامه کل جواب را به نمایش گذاشته باشد، به سطر بعد رفته و دوباره یک علامت (؟\_) می‌گذارد در غیراینصورت باید کلید space را بزنید تا بقیه جواب‌ها ظاهر شود. و یک نکته دیگر این است که اگر X و Y یا .... را با حروف کوچک بنویسید، برنامه FALS می‌دهد.

```

File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 7.1.10)
Copyright (c) 1990-2014 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- parent(x, y).
ERROR: toplevel: Undefined procedure: parent/2 (DWIM could not correct goal)
2 ?- PARENT(X, Y).
ERROR: Syntax error: Operator expected
ERROR: PARENT
ERROR: ** here **
ERROR: (X, Y) .
2 ?- parent(X, Y).
ERROR: toplevel: Undefined procedure: parent/2 (DWIM could not correct goal)
3 ?-
% C:/Users/ASH/Desktop/New Text Document (4).pl compiled 0.00 sec, 4 clauses
3 ?- parent(X, Y).
X = jim.
Y = tom.

4 ?- █

```

نمونه‌ای از برنامه کد شده