پردیس علوم
دانشکده ریاضی، آمار و علوم کامپیوتر

آشنایی با زبان جولیا (Julia)

نگارنده

سارینا سفیدگر حسینی

استاد راهنما: مهندس ابراهیم نقیب زاده مشایخ

پروژه برای دریافت درجه کارشناسی
در رشته علوم کامپیوتر

تیر ۱۳۹۸

چکیده

در این مقاله قصد معرفی زبان برنامه‌نویسی جولیا را داریم. این زبان را با دیگر زبان‌های برنامه‌نویسی پرکاربرد مقایسه می‌کنیم تا علت رشد کردن این زبان و محبوب شدنش را نشان دهیم. این زبان در سال‌های آتی بسیار مورد توجه قرار خواهد گرفت و جایگزین زبان‌های پرکاربردی مانند پایتون[1] و آر[2] خواهد شد. ویژگی‌های این زبان را بیان خواهیم کرد تا شما را با این زبان بسیار پرکاربرد آشنا کنیم.

به طور خلاصه جولیا را آموزش می‌دهیم. این زبان بسیار گسترده است و برای یادگیری به منابعی بیشتر از یک مقاله نیاز است. اما تمام تلاش‌مان را خواهیم کرد تا کاربردهای مهم را در اختیار شما قرار دهیم.

کلمات کلیدی فارسی: جولیا، پایتون، آر، برنامه‌نویسی پویا، اجرای موازی، متن باز، ارسال چندگانه، سرعت بالا، نوع اختیاری، اخبار جولیا، اهداف جولیا، جولیا در مقابل پایتون و آموزش زبان جولیا.

keywords: Julia , Python , Parallel , Dynamic Programming , R , Open Source , Multiple Sending , High Speed , Optional Type , Julia's News , Julia's Goals , Julia vs. Python , Julia learning.

[1] python
[2] R

پیشگفتار

زبان برنامه‌نویسی جولیا، یکی از جدیدترین زبان‌هایی است که توسط دانشگاه MIT توسعه یافته و برخی آن را زبان برنامه‌نویسی محبوب آینده می‌دانند.

چرا جولیا؟

جولیا یک زبان برنامه‌نویسی پویا با سطح بالا و با کارایی بالا برای محاسبات عددی است. این زبان یک کامپایلر پیشرفته، اجرای هم‌زمان موازی، دقت عددی و یک کتاب‌خانه تابع گسترده ریاضی را فراهم می‌کند.

کتاب‌خانه پایه جولیا(که عمدتا در جولیا نوشته شده است) هم‌چنین کتاب‌خانه‌های متن باز زبان سی و فورترن را برای جبر خطی، تولید عدد تصادفی، پردازش سیگنال و پردازش رشته با هم ترکیب می‌کند. مهم‌تر از همه رایگان و متن باز بودن آن با این همه ویژگی است. [۷]

جولیا در مقایسه با دیگر زبان‌های برنامه‌نویسی؟

ساده‌ترین راه برای درک برتری زبان جولیا این است که بدانید:

- دارای طیف گسترده‌ای از بسته‌های آماری است. مانند زبان R.
- آسان یاد بگیرید و آسان برنامه‌نویسی کنید. مانند زبان پایتون.
- سرعت اجرای بالا مشابه آن‌چه در زبان C و C++ شاهد آن هستیم.

هرچند جولیا هنوز جایی بین ۱۰ زبان برنامه‌نویسی محبوب جهان ندارد، اما دو رده‌بندی RedMonk و TIOBE آن را جزو زبان‌های برنامه‌نویسی قرار داده که با سرعت بسیار بالایی توسط تولیدکنندگان مورد استفاده قرار گرفته و استفاده از آن گسترش می‌یابد.[۳]

در فصل اول مفاهیم مقدماتی جولیا بیان خواهد شد. در مورد کاربردهای جولیا و چگونگی نصب آن و ویژگی‌های آن توضیح داده خواهد شد.

در فصل دوم جولیا را با سایر زبان‌های پرکاربرد مقایسه می‌کنیم. در فصل سوم هم به طور خلاصه به آموزش زبان جولیا می‌پردازیم.

چند ویدئوی مناسب برای آموزش جولیا به شرح زیر است:

https://www.youtube.com/watch?v=8h8rQyEpiZA&t
https://youtu.be/SLE0vz85Rqo
https://www.youtube.com/watch?v=OFPNph-WxLM
https://www.youtube.com/watch?v=0-Xsts_s5W
https://www.youtube.com/watch?v=13hqE_1a158
https://youtu.be/LbTbs-0pOuc

چند نوشته مناسب:

https://benlauwens.github.io/ThinkJulia.jl/latest/book.html
http://ucidatascienceinitiative.github.io/IntroToJulia/
https://github.com/bkamins/The-Julia-Express
https://lectures.quantecon.org/jl/
https://en.wikibooks.org/wiki/Introducing_Julia
https://www.analyticsvidhya.com/blog/2017/10/comprehensive-tutorial-learn-data

چند کتاب مناسب:

http://vmls-book.stanford.edu/
https://www.amazon.com/Algorithms-Optimization-Press-Mykel-Kochenderfer/dp/0262039427/
http://www.chkwon.net/julia
http://shop.oreilly.com/product/0636920215707.do
https://www.crcpress.com/Data-Science-with-Julia/McNicholas-Tait/p/book/9781138499980
https://www.packtpub.com/application-development/mastering-julia
https://www.packtpub.com/application-development/getting-started-julia-program
https://www.packtpub.com/big-data-and-business-intelligence/julia-data-science
https://www.packtpub.com/big-data-and-business-intelligence/julia-solutions-video
https://www.packtpub.com/application-development/getting-started-julia-video

فهرست مطالب

فصل ۱

مفاهیم مقدماتی

در این فصل با داستان شکل‌گیری زبان برنامه‌نویسی جولیا، انواع کاربردهای مهم زیان جولیا، ویژگی‌های مهم آن، چگونگی نصب جولیا و بسته‌های آن آشنا خواهیم شد. هم‌چنین زیست بوم جولیا را نیز بیان خواهیم کرد.

۱.۱ داستان شکل‌گیری زبان برنامه‌نویسی جولیا

جولیا هم‌چنان در ابتدای راه قرار دارد و در میان قدرتمندترین و پرنیازترین زبان‌های برنامه‌نویسی به حساب نمی‌آید، ولی اشاره به این دو نکته درک خوبی از محبوبیت روزافزون این زبان برنامه‌نویسی فعلاً گمنام به شما می‌دهد: اول از همه این‌که در همین ابتدای کار کاربران جولیا از مرز ۲۰۰ هزار نفر گذشته‌اند و نکتهٔ دوم هم در نرخ رشد این زبان برنامه‌نویسی نهفته است؛ نرخ رشد این زبان برنامه‌نویسی در هر ۹ ماه، چیزی در حدود ۲ برابر می‌شود.

در ادامه بخشی از مصاحبه‌ای که با ویرال شاه[1]، یکی از مؤسسین Computing julia در مورد تکامل و وضعیت سرعت رشد این زبان برنامه‌نویسی را با هم مرور می‌کنیم.

ما این پروژه را در سال ۲۰۰۹ آغاز کردیم که بعداً تبدیل به زبان برنامه‌نویسی جولیا شد. من در رشتهٔ علوم کامپیوتر تحصیل کردم.

پایان‌نامهٔ من در دانشگاه کالیفرنیا بر روی محاسبات موازی متمرکز بود و بعدتر بخشی از محصول Star-P در شرکت Interactive Supercomputing Corp شد. زمانی که مایکروسافت این شرکت را خریداری کرد، من و جف بزانسون [2](دیگر مؤسس شرکت) کار خود را در مورد روش‌های جدید محاسبات موازی آغاز کردیم. به طور هم‌زمان هم من و استفان کارپینسکی [3](دیگر

[1] Viral Shah
[2] Jeff Bezanson
[3] Stefan Karpinski

مؤسس شرکت) در مورد مشکلات مشابهی که در همکاری تحقیقاتی‌مان در UCSB داشتیم صحبت می‌کردیم. پس از آن، جف به آلن ادلمن [۴] (یکی دیگر از مؤسسین شرکت) که در پایان‌نامه‌ام همکاری داشت پیشنهاد پیوستن به ادامهٔ تحصیلات در مقطع دکتری در MIT را ارائه کرد و به این صورت ما ۴ نفر یک‌جا جمع شدیم.

ویرال شاه در ادامهٔ داستان شکل‌گیری زبان برنامه‌نویسی جولیا این‌گونه می‌گوید که:

ما در سال ۲۰۰۹ موفق شدیم مشکل «دو زبانی» را حل کنیم. مهم‌ترین چیزی که مانع پیشرفت ما در محاسبات موازی می‌شد این حقیقت بود که کاربرانی که از زبان‌های برنامه‌نویسی سطح بالایی مانند R و Python استفاده می‌کنند باید بخش‌های اجرایی را در C یا C++ بازنویسی کنند. این مشکل به شدت باعث پایین آمدن کارایی می‌شد چرا که این کار همیشه موجب به وجود آمدن خطاهای انسانی، از بین رفتن وقت و تلاش، کاهش سرعت فروش و در نهایت عقب افتادن از رقبا می‌شد. این مشکل دو زبانی نه تنها در کار محققین، بلکه در کار دانشمندان، داده پژوهان، مهندسین و تحلیل‌گران مالی موانع زیادی ایجاد می‌کرد.

این بدان معنی است که اکنون جولیا توجه تمام کسانی که در فعالیت‌های اقتصادی شرکت دارند را به خود جلب کرده است. همهٔ این افراد، از تجار گرفته تا مدیران اجرایی، تولیدکنندگان، داده پژوهان و خلاصه هر کسی که قصد دارد مهارت‌های خود را در این دنیای مبتنی بر تجارت الکترونیکی امروز ارتقاء دهد، توجه ویژه‌ای به زبان برنامه‌نویسی جولیا دارند. [۴]

۲.۱ کاربردهای جولیا

جولیا قابلیت‌های متنوع و متعددی دارد و همین امر جولیا را به گزینه‌ی مناسبی برای استفاده در کاربردهای گوناگون تبدیل کرده است. برای مثال جولیا این امکان را دارد تا بتوان با استفاده از آن وظایف پردازشی را بین هسته‌های مختلف تقسیم کرد که این ویژگی امکان استفاده از جولیا از یادگیری ماشین تا شبیه‌سازی ابررایانه‌های بزرگ را فراهم می‌کند.

MIT اعلام کرده که زبان برنامه‌نویسی جولیا را باید پویاترین زبان برنامه‌نویسی در باشگاه زبان‌های برنامه‌نویسی موسوم به پتافلاپ خواند. باشگاه پتافلاپ به گروهی از زبان‌های برنامه‌نویسی اطلاق می‌شود که با استفاده از آن‌ها می‌توان از قدرت پردازشی یک پتافلاپ در ثانیه عبور کرد. براساس اطلاعات ارائه‌شده، محققان با استفاده از جولیا اقدام به شبیه‌سازی ۱۸۸ میلیون ستاره و کهکشان در ابر رایانه Cori کرده‌اند که دهمین ابررایانه‌ی قدرتمند در جهان است. این شبیه‌سازی در کمتر از ۱۵ دقیقه انجام شده است. در این شبیه‌سازی، بیش از ۶۵۰ هزار هسته‌ی پردازشی Knights Landing Xeon Phi مورد استفاده قرار گرفته، که نتیجه‌ی آن قدرت پردازشی ۵.۱ پتافلاپ بوده است.

از جمله‌ی دیگر کاربردهای جولیا می‌توان به استفاده از آن در خودروهای خودران و چاپگرهای سه بعدی در کنار پزشکی و لوازم پزشکی با دقت بالا، واقعیت افزوده، ژنتیک، یادگیری ماشین و

[4] Alan Edelman

مدیریت ریسک اشاره کرد.
از جمله‌ی سیستم‌های توسعه‌یافته با استفاده از جولیا باید به توسعه‌ی نسل بعدی سیستم جلوگیری از برخورد هواپیما، بهبود مسیریابی اتوبوس مدارس بوستون و هم‌چنین سیستم حرکتی و مسیریابی ربات اشاره کرد.
جولیا توسط آزمایشگاه هوش مصنوعی و علم رایانه‌ی دانشگاه MIT توسعه یافته است. این زبان برنامه‌نویسی متن‌باز و رایگان بوده و بیش از ۹۰۰۱ پکیج ثبت شده دارد. هم‌چنین باید به دو میلیون بارگیری در کنار افزایش ۱۰۱ درصدی رشد بارگیری سالیانه‌ی این زبان برنامه‌نویسی اشاره کرد.[۴]

۳.۱ خلاصه‌ای از ویژگی‌های جولیا

برخی از ویژگی‌های برجسته جولیا در کاربرد علوم داده:

- ارسال چندگانه: ارائه توانایی برای تعریف رفتار تابع در بین بسیاری از ترکیبات با انواع استدلال.
- سیستم پویا: الگو برای اسناد، بهینه سازی و ارسال.
- کارایی خوب، نزدیکی به زبان‌های کامپایل شدنی نظیر C
- مدیریت بسته درون‌ـساخت، کار را آسان‌تر می‌کند.
- دارای امکانات Meta-Programming به معنای طراحی اپلیکیشن بهتر، سریع‌تر و با حجم کد کمتر می‌باشد.
- فراخوانی توابع C به طور مستقیم.
- فراخوانی توابع پایتون با استفاده از بسته PyCall.
- قابلیت قدرتمند خط فرمان برای مدیریت فرایندهای دیگر.
- طراحی شده برای موازی سازی و محاسبات توزیع شده.
- همان‌قدر که تعاریف داخلی سریع و جمع و جور هستند، تعاریف کاربر هم همین‌گونه هستند.
- تولید خودکار کد کارآمد و ویژه برای انواع استدلال مختلف.
- زیبا و گسترش‌پذیر و قابل ترویج برای داده‌های عددی و انواع دیگر.

- مجوز :MIT منبع آزاد و رایگان.

۱.۳.۱ ویژگی‌های منحصر به فرد جولیا

- سرعت بالا:
Julia از ابتدا برای عملکرد بالا و سرعت زیاد طراحی شده بود. برنامه‌های Julia با استفاده از کد محلی کارآمد برای چند سیستم عامل از طریق LLVM کامپایل می‌شوند.

- پویا:
Julia به صورت پویا تایپ شده و به عنوان یک زبان برنامه‌نویسی، پشتیبانی خوبی برای استفاده تعاملی دارد.

- نوع اختیاری:
Julia دارای زبان غنی از انواع داده‌های توصیفی است و از انواع اعلامیه‌ها برای روشن شدن و جامعیت برنامه‌ها استفاده می‌کند.

- عمومی:
Julia از چندین اعلان به عنوان یک پارادایم استفاده می‌کند، به طوری که بسیاری از الگوهای برنامه‌نویسی شیءگرا و کاربردی را بیان می‌کند.

- فنی:
این زبان هم‌چنین در محاسبات عددی در سطح بالایی قرار دارد. نحو Julia برای ریاضی بسیار عالی است و از انواع داده های عددی پشتیبانی می‌کند.

- قابلیت خواندن:
بسته‌های جولیا به طور طبیعی با هم کار می‌کنند. این بسته‌ها شامل کتاب‌خانه‌های ریاضی، ابزار دست‌کاری داده‌ها و بسته‌های رایانه‌ای عمومی می‌باشند. علاوه بر این شما می‌توانید از کتاب‌خانه‌های C و پایتون و جاوا نیز استفاده کنید.[۳]

۴.۱ نصب جولیا

حالا اگه با تفاسیر بالا می‌خواهید با این زبان کار آمد فعالیت کنید، با ما همراه باشید تا خیلی سریع، راه‌های کار با زبان جولیا رو بیان کنیم:

راه اول: استفاده از juliabox در مرورگرتان.

ساده‌ترین راه همین است که بدون نصب و فقط با استفاده از حساب گوگل خودتان وارد سایت زیر بشوید و شروع کنید به برنامه‌نویسی.

```
http://www.juliabox.org
```

راه دوم: استفاده از یک محیط توسعه مجتمع.
در حال حاضر بهترین محیط توسعه مجتمع برای جولیا، Juno هست.
راه سوم: استفاده از خط دستور
اگه شما برنامه‌نویسی حرفه‌ای هستید و بدون خط فرمان نمی‌توانید برنامه‌نویسی را تصور کنید، بسته خط فرمان جولیا را از لینک زیر بارگیری کنید.[۸]

```
http://julialang.org/downloads/
```

۵.۱ تعدادی از بسته‌های مهم جولیا

مجموعه‌ای از ۶۱۰ بسته جولیا در تاریخ(۹ ژوئیه ۲۰۱۵) وجود داشت. اگر شما بسته‌هایی را که در آزمایشات شکست خورده یا آزمایش نشده‌اند فیلتر کنید، تنها ۳۸۱ بسته باقی می‌ماند. در میان این‌ها، موارد مرتبط با علم داده را فیلتر کرده‌ایم که دارای بیش از ۱۵ ستاره‌اند. نتیجه بسته‌های زیر هستند:[۹]

بسته	توضیحات	نسخه	ستاره
Mocha	Deep Learning framework for Julia	0.0.8	297
MultivariateSta	A Julia package for multivariate statistics , data analysis	0.2.1	21
NLopt	Package to call the NLopt nonlinear-optimization library	0.2.1	31
OpenStreetMa	Julia OpenStreetMap Package	0.8.1	20
Optim	Optimization functions for Julia	0.4.2	116
Orchestra	Heterogeneous ensemble learning for Julia	0.0.5	27
PGM	A Julia framework for probabilistic graphical models	0.0.1	25
PyCall	Package to call Python functions	0.8.1	183
RCall	Embedded R within Julia	0.2.1	16
RDatasetsl	Julia package for loading many of the data sets available in R	0.1.2	34
Regression	Algorithms for regression	0.3.2	17
Rif	Julia-to-R interface	0.0.12	47
StatsBase	Basic statistics for Julia	0.6.15	57
StreamStats	Compute statistics over data streams in pure Julia	0.0.2	27
TimeSeries	Time series toolkit for Julia	0.5.10	37

توجه کنید که:

ستاره	نسخه	توضیحات	ته
18	0.0.3	A neural network in Julia	Backprop
26	0.1.0	Bokeh Bindings for Julia	Bo
19	0.1.0	Restricted Boltzmann Machines in Julia	Boltz
46	0.1.8	Calculus functions in Julia	Calc
33	0.4.0	A Julia package for data clustering	Clust
108	0.0.6	A julia package for disciplined convex programming	Con
18	0.1.0	Utilities for calling C++ from Julia	Cp
21	0.2.16	Data structures that allow missing values	DataA
206	0.6.7	library for working with tabular data in Julia	DataF
33	0.0.1	Metaprogramming tools for DataFrames	DataFra
52	0.3.10	Julia implementation of Data structures	DataStr
36	0.3.8	Decision Tree Classifier and Regressor	Decisi
21	0.2.0	A package for evaluating distances(metrics) between vectors	Dista
101	0.7.4	A package for probability distributions, associated functions	Distrib
32	0.0.8	Filter design, periodograms, window functions	D
34	0.1.2	Functional and and persistent data structures for Julia	Functional
684	0.3.13	Crafty statistical graphics for Julia	Gad
86	0.0.3	A lightweight framework for writing genetic algorithms in Julia	GeneticA
78	0.4.6	Generalized linear models in Julia	GL
23	0.0.4	Wrapper for fitting Lasso/ElasticNet GLM models using glmnet	GLM
90	0.5.5	Working with graphs in Julia	Gra
65	0.4.18	Saving and loading Julia variables	HD
16	0.2.9	Hypothesis tests for Julia	Hypothe
73	0.4.39	An image library for Julia	Ima
162	0.9.2	Modeling language for Mathematical Programming	Jul
37	0.0.3	Julia Machine Learning library	Machine
44	0.4.11	Markov chain Monte Carlo (MCMC) for Bayesian analysis in julia	Mar
21	0.3.0	Markdown parsing for Julia	Mark
29	0.1.3	Advanced Pattern Matching for Julia	Ma
41	0.3.22	A Julia package for fitting mixed-effects models	Mixed
41	0.5.1	A set of functions to support the development of machine learning	ML

- Gadfly به نظر می‌رسد محبوب‌ترین بسته است. این شاید به خاطر آن است که به عنوان یک کتاب‌خانه ویترین در تمام محصولات موجود در زیست بوم استفاده می‌شود.

- کتاب‌خانه‌های وابسته به علوم داده نسبت به برخی از کتاب‌خانه‌ها بیشتر تکامل یافته‌اند. Mocha برای یادگیری عمیق، Orchestra برای بهینه سازی و DataFrames یا Distributions در برابر دیگر کتاب‌خانه‌ها تکامل یافته‌تر هستند.

۱.۵.۱ چگونه یک بسته را نصب و استفاده کنیم؟

این کار در جولیا به سادگی آب خوردن است! برای نصب/اضافه کردن یک بسته کافیست کد دستوری زیر را بنویسید:

```
Pkg.add("Gadfly")
```

این دستور، بسته و وابستگی‌های آن را برای شما نصب/اضافه می‌کند.
یک بار که بسته را نصب کردید، با کد زیر می‌توانید آن را صدا بزنید:

```
using Gadfly
```

به همین سادگی!

۶.۱ زیست بوم جولیا

جولیا توسط یک جامعه نزدیک از توسعه‌دهندگان پشتیبانی می‌شود. در زیر لیستی از این پشتیبانی‌ها آورده شده است:

- اخبار جولیا: برای اطلاعیه‌های مهم مانند نسخه جدید.

- کاربران جولیا: بحث در مورد استفاده از جولیا. کاربران جدید جولیا می‌توانند سؤالات خود را در این‌جا مطرح کنند.

- julia-stats: لیستی مخصوص با هدف بحث در مورد برنامه‌های آماری با جولیا. موضوعات مورد علاقه شامل پشتیبانی از DataFrame، مدل سازی GLM و نسل کدهای MCMC برای مدل های Bayesian است.

- julia-opt: بحث در مورد بهینه‌سازی عددی در جولیا. این شامل برنامه‌نویسی ریاضی (خطی، عدد صحیح مخلوط، مخروطی، نیمه قطعی، و غیره)، محدودیت و بدون محدودیت بهینه‌سازی مبتنی بر شیب و بدون گرادیان و موضوعات مرتبط است.

علاوه بر این، شما هم‌چنین می‌توانید به سایت زیر نگاه کنید. این سایت به نظر می‌رسد مانند یک زیست بوم در حال توسعه است.[۱۰]

`http://www.juliabloggers.com`

فصل ۲

مبارزه نفس‌گیر میان زبان برنامه‌نویسی جولیا و پایتون در حوزه علم داده‌ها

به جرات می‌توان ادعا کرد که در زمینه علم داده‌ها و یادگیری ماشین پایتون قدرت بلامنازع است. اما زبان برنامه‌نویسی جولیا به آرامی در حال قدرت گرفتن و فراگیر شدن است. این زبان آماده است تا در زمینه علم داده‌ها و بسیاری دیگر از ویژگی‌ها پایتون را به چالش بکشد.

از کاربردهای گسترده پایتون به تجزیه و تحلیل داده‌ها می‌توان اشاره کرد که یکی از بزرگ‌ترین و مهم‌ترین قابلیت‌های پایتون به شمار می‌رود. زیست بوم پایتون همراه با مجموعه غنی از کتابخانه‌ها، ابزارها و برنامه‌های کاربردی به خوبی قادر است در زمینه محاسبات علمی و تجزیه و تحلیل داده‌ها به کار گرفته شود. اما توسعه‌دهندگان که در پس‌زمینه زبان جولیا قرار دارند به دنبال هدف دیگری هستند. آن‌ها به دنبال آن هستند تا زبان جولیا را به شکل خاص منظور برای محاسبات علمی، یادگیری ماشین، داده‌کاوی، جبر خطی در مقیاس بزرگ، محاسبات توزیع‌شده و هم‌چنین محاسبات موازی آماده کنند.

در حالی که پایتون در برخی از این بخش‌ها خیلی خوب عمل می‌کند، اما در مقابل در بخش‌های دیگر مشکل جدی دارد، سریع نبودن یکی از نقاط ضعف آشکار پایتون است.[۵]

۱.۲ زبان جولیا چیست؟

جولیا در سال ۲۰۰۹ میلادی از سوی یک تیم چهار نفره ساخته شد و در سال ۲۰۱۲ میلادی به شکل عمومی در اختیار مردم قرار گرفت. جولیا آمده است تا نقاط ضعف موجود در پایتون و زبان‌های برنامه‌نویسی را برطرف کرده و در زمینه محاسبات علمی و پردازش داده‌ها نیز حضور پر رنگی داشته باشد. تیم سازنده در این ارتباط گفته است: «ما حریص هستیم، ما زبانی می‌خواهیم که متن‌باز باشد و تحت یک مجوز عمومی در اختیار عموم مردم قرار گیرد. ما می‌خواهیم سرعت زبان

C را با پویایی زبان روبی ادغام کنیم. ما به دنبال هماهنگ کردن جولیا با تابع‌های واقعی هستیم، درست همانند رویکردی که در زبان لیسپ شاهد آن هستیم. ما در نظر داریم به کاربران اجازه دهیم با نشانه‌ها و فرمول‌های ریاضی درست همانند زبان متلب کار کنند. ما می‌خواهیم تا مولفه‌های این زبان همانند زبان پایتون در زمینه برنامه‌نویسی عمومی به شکل قابل قبولی به کار گرفته شوند. ما به دنبال آن هستیم تا اجازه دهیم شما از جولیا در کارهای آماری درست همانند زبان آر استفاده کرده و در زمینه پردازش طبیعی به خوبی پرل عمل کرده و در زمینه جبر خطی همان قدرت متلب را داشته باشید. زبانی که یادگیری آن تا حد امکان ساده باشد. ما یک زبان تعاملی می‌خواهیم که بتوانیم آن‌را کامپایل کنیم.»[۵]

۲.۲ جولیا به دنبال چه اهدافی است؟

جولیا برای آن‌که سرعت محاسبات را افزایش دهد از مکانیزم کامپایل به جای تفسیر استفاده می‌کند. جولیا از کامپایلر LLVM استفاده می‌کند. در بهترین حالت، جولیا می‌تواند به لحاظ سرعت با زبان C برابری کرده یا به آن نزدیک شود.

جولیا در نظر دارد یک ترکیب نحوی ساده اما کاربردی را ارائه کند. ترکیب نحوی جولیا شباهت زیادی به پایتون دارد. اما در بعضی زمینه‌ها قدرت‌مندتر از پایتون است.

جولیا به دنبال آن است تا از مزایای نوع‌های ایستا و پویا به یک اندازه بهره‌مند شود. شما می‌توانید برای متغیرها نوع‌هایی شبیه به unsigned 32-bit integer را تعریف کنید. اما هم‌چنین می‌توانید سلسله مراتبی از نوع‌ها را ایجاد کرده که برای کارهای عمومی و به منظور مدیریت متغیرها در ارتباط با نوع‌های خاصی به کار گرفته شوند. به‌طور مثال تعریف تابعی که می‌تواند مقادیر صحیحی را بدون مشخص کردن طول یا نوع دریافت کند. هم‌چنین می‌توانید بدون تعریف صریح نوع متغیر که از خصایص نوع‌های پویا است در این زبان استفاده کنید.

کتاب‌خانه‌های پایتون، C و فورترن تنها از طریق یک فراخوانی در دسترس هستند. جولیا می‌تواند به‌طور مستقیم با کتاب‌خانه‌های خارجی که به زبان C یا فورترن نوشته شده‌اند ارتباط برقرار کند. هم‌چنین از طریق کتاب‌خانه PyCall قادر است از کدهای پایتون پشتیبانی کرده و حتی الامکان به‌اشتراک‌گذاری داده‌ها میان پایتون و جولیا نیز وجود دارد.

ساخت‌یافته بودن یکی از مهم‌ترین خصایص جولیا است. شبیه به زبان برنامه‌نویسی لیسپ در زبان برنامه‌نویسی جولیا کدهای برنامه به عنوان یک ساختار داده‌ای شناخته می‌شوند.[۵]

۳.۲ جولیا در مقابل پایتون

جولیا از ابتدا به گونه‌ای طراحی شده که در زمینه محاسبات علمی و آماری به کار گرفته شود. بنابراین جای تعجب نیست که جولیا ویژگی‌های مثبت زیر را داشته باشد.

- جولیا به شکل پیش‌فرض سریع است.
کامپایلر جولیا و شیوه تعریف متغیرها در این زبان به شکل قابل توجهی سریع‌تر از پایتون است. پایتون می‌تواند از طریق به کارگیری کتاب‌خانه‌های خارجی، کامپایلرهای JIT ثالث (PyPy) سریع‌تر شود و هم‌چنین از طریق ابزارهایی شبیه به Python بهینه‌سازی شود، اما جولیا به‌گونه‌ای طراحی شده که از همان ابتدا این ویژگی را داشته باشد.

- یک ترکیب نحوی متناسب با ریاضیات
مخاطبان اصلی جولیا کاربرانی هستند که عمدتا با زبان‌های محاسباتی علمی و محیط‌هایی شبیه به متلب، آر، Mathematica و Octave سروکار دارند. ترکیب نحوی و محیط جولیا برای عملیات ریاضی هم‌چون فرمول‌های ریاضی که در دنیای واقعی از آن‌ها استفاده می‌کنیم کاملا ایده‌آل است. در نتیجه جولیا برای افرادی که برنامه‌نویس به شمار نمی‌روند یک محیط ساده را فراهم آورده است.

- مدیریت خودکار حافظه
شبیه به پایتون، جولیا کاربر را درگیر جزییات مربوط به تخصیص و آزادسازی حافظه نمی‌کند و همچنین یکسری کنترل‌های دم‌دستی به منظور مدیریت بر زباله روبی در اختیار کاربر قرار می‌دهد. در نتیجه اگر از پایتون به جولیا بروید، در عمل شما با مشکل خاصی روبرو نمی‌شوید و با مفاهیم مشترکی سروکار خواهید داشت.

- عملیات موازی
جولیا می‌تواند عملیات را به شکل موازی انجام دهد. اما ترکیب نحوی جولیا در این زمینه کمی از پایتون سنگین‌تر بوده و در نتیجه به نظارت بیشتری نیاز دارد.

اما فراموش نکنید که آرایه‌ها در جولیا با اندیس ۱ آغاز می‌شوند. جولیا هنوز هم یک زبان خیلی جوان است. جولیا در مقایسه با پایتون بسته بندی‌های بخش ثالث بیشتری در اختیار داشته و در نهایت پایتون جامعه بسیار وسیعی از توسعه‌دهندگان را در اختیار دارد.[۵]

۴.۲ مقایسه سرعت اجرا توابع منتخب نسبت به زبان C

در جدول زیر مقایسه ها را انجام داده‌ایم.
اعداد هر چه کمتر باشد بهتر است.[۱۱]

Fortran	Julia	Python	R	Matlab	Mathematica	JavaScript	Java	Function
0.57	2.14	95.45	528.85	4258.12	166.64	3.68	0.96	Fib
4.67	1.57	20.48	54.30	1525.88	17.70	2.29	5.43	ParseInt
1.10	1.21	46.70	248.28	55.87	48.47	2.91	1.65	QuickSort
0.87	0.87	18.83	58.97	60.09	6.12	1.86	0.68	Mandel
0.83	1.00	21.07	14.45	1.28	1.27	2.15	1.00	PiSum
0.99	1.74	22.29	16.88	9.82	6.20	2.81	4.01	RandMatStat
4.05	1.09	1.08	1.63	1.12	1.13	14.58	2.35	RandMatMul

فصل ۳

آموزش زبان برنامه‌نویسی جولیا

جولیا یک زبان جدید کامپیوتری با تمرکز بر محاسبات عددی می‌باشد. این زبان مانند زبان لیسپ "Homoiconic" است، به این معنا که کدهای برنامه نیز داده‌هایی از برنامه به شمار می‌روند، که امکان تولید کدهای برنامه‌نویسی پویا [۱]را به برنامه‌نویس می‌دهد.

در تعریف توابع بسیار قوی و انعطاف‌پذیر می‌باشد، که امکان تعریف رویه‌ها[۲] و عملگرهای محاسباتی جامع[۳] برای ساختارهای داده متفاوت را فراهم می‌آورد.

در این فصل به طور خلاصه زبان برنامه‌نویسی جولیا را برای مبتدی‌ها آموزش خواهیم داد.

۱.۳ شروع کار با جولیا

در شروع کار با جولیا با ما همراه باشید. همان‌طور که در فصل پیش در نصب جولیا گفته شد، نصب این زبان برنامه‌نویسی خیلی ساده است و شما به راحتی می‌توانید آن را برای هر سیستمی(سخت افزاری و سیستم عاملی) از روی کد منبع آن در https://julialang.org/downloads نصب کنید.

ساده‌ترین راه برای یادگیری جولیا، اجرای خود جولیا REPL[۴]است. برای اجرای آن یا بر روی آن دوبار کلیک کنید و یا از خط فرمان سیستم julia را صدا بزنید.

برای خروج از محیط تعاملی می‌توانید از ترکیب CTRL-D استفاده کنید یا فرمان ()exit را وارد کنید.

جولیا پس از اجرا شدن در محیط تعاملی یک بنر را نمایش می‌دهد و در پایین آن منتظر ورودی کاربر می شود. زمانی که کاربر یک دستور کامل را به جولیا بدهد(مثل 1+2) سپس کلید enter

[1] Meta-Programming
[2] Methods
[3] Generic
[4] Read-Eval-Print-Loop

را فشار دهد، محیط تعاملی آن را بررسی کرده و ارزش آن را نمایش می‌دهد.
اگر در محیط تعاملی پس از دستور خود سمیکالن (؛) بگذارید و کلید enter را بفشارید، ارزش کد شما(خروجی بیان) نمایش داده نمی‌شود.
متغیر ans ارزش آخرین بیان(دستور) شما را نشان می‌دهد و فرقی ندارد که قبلا نمایش داده شده یا از نمایش آن بوسیله (؛) جلوگیری شده است.
متغیر ans فقط در محیط تعاملی (interactive sessions) کار می‌کند و در صورت‌های دیگری که کد جولیا را اجرا می‌کنید کارایی ندارد.
برای ارزیابی عملکرد یک کد منبع در فایلی به مشخصات file.jl باید include("file.jl") را بنویسید(فرمان دهید). توجه داشته باشید که این فایل باید در پوشه‌ای که جولیا را نصب کرده‌اید موجود باشد.
اگر می‌خواهید کد درون فایل را به صورت غیر تعاملی اجرا کنید باید در خط فرمان خود فایل را به عنوان اولین نشانوند(ورودی) دستور julia قرار دهید(توجه کنید که در خط فرمان ویندوز به جای(') باید از(") استفاده کنید).

julia script.jl arg1 arg2...

همان‌طور که در مثال بالا مشاهده می‌کنید، خط فرمان نوشته شده نشانوندهای بعد از نشانوند فایل را به صورت خط فرمان تفسیر کرده و آن‌ها را به script.jl می‌فرستد که به ثابت جهانی ARGS پاس داده می‌شوند.
نام سند نیز خودش به شاخص جهانی PROGRAM_FILE پاس داده می‌شود.
توجه داشته باشید زمانی که اصطلاح julia در خط فرمان عبارت e- را می‌گیرد نیز ARGS در کار وجود دارد اما خبری از PROGRAM_FILE نیست.
برای آشنایی بیشتر با فرمان‌های خط فرمان برای جولیا کافیست h- را به جولیا در خط فرمان بدهید.
به عنوان مثال، برای فقط چاپ نشانوند داده شده به یک سند، شما می‌توانید این کار را انجام دهید:

```
julia -e 'println(PROGRAM_FILE); for x in ARGS; println(x); end' foo bar
foo
bar
```

یا می‌توانید کد را درون متن سند قرار داده سپس آن را اجرا کنید:

```
echo 'println(PROGRAM_FILE); for x in ARGS; println(x); end' > script.jl
julia script.jl foo bar
script.jl
foo
bar
```

نماد – می‌تواند میان نشانوندهای اختصاص داده شده به سند و نشانوندهای اختصاص داده شده به جولیا تفاوت ایجاد کند. نماد – نشانوندها را به جولیا می‌فرستد:

julia –color=yes -O – foo.jl arg1 arg2..

جولیا با هر دو دستور انتخابی p- یا machine-file– می‌تواند در حالت پردازش موازی [۵] شروع به کار کند.

p n- به اندازه n عملیات‌گر پردازش را اجرا می‌کند در حالی که machine-file file– به ازای هر خط از فایل یک عملیات‌گر را اجرا می‌کند.

راه‌های مختلفی برای اجرای کد جولیا وجود دارد و شبیه گزینه‌هایی هستند که برای برنامه‌های پرل[۶] و روبی[۷] ارائه شده‌اند:[۱۲، ۵، ۴]

julia [switches] – [program_file] [args...]

۲.۳ متغیرها در جولیا

حال که با این زبان قدرت‌مند و ساده آشنا شدید و به کمک فصل پیش آن را نصب کرده و از طریق شروع کار با جولیا آن را اجرا کرده و آماده‌اید برای یادگیری این زبان! بگذارید با متغیرها در جولیا و قوانین نام‌گذاری آن‌ها کمی آشنا شویم. چرا که اگر متغیرها نباشند عملاً برنامه‌نویسی هم نیست! متغیرها در جولیا چیستند؟ یک متغیر در جولیا(و کلاً در برنامه‌نویسی) نامی است که به یک مقدار اختصاص می‌دهیم یا به عبارتی قسمتی از حافظه سیستم است که با یک نام نشانه‌گذاری شده و مقدار مشخصی را در خود جای می‌دهد. این کار برای این مفید است که بخواهید به مقدار مشخصی در حافظه ذخیره شده دسترسی پیدا کنید و آن را تغییر داده یا با آن عملیاتی را انجام دهید.

به عنوان مثال به کد زیر نگاهی بیاندازید:

اختصاص مقدار ۱۰ به متغیر x

```
julia> x = 10
10
```

انجام عملیات ریاضی به وسیله مقدار ریخته شده درون متغیر x

```
julia> x + 1
11
```

اختصاص‌دهی دوباره به متغیر x

[5] parallel
[6] perl
[7] ruby

```
julia> x = 1 + 1
2
```

شما هر نوع داده‌ای را می‌توانید به متغیرتان تخصیص دهید و برای همین به جای کلمه "مقدار" از کلمه "ارزش" استفاده می‌کنیم.

```
julia> x = "Hello World!"
"Hello World!"
```

متغیرها در جولیا یک سیستم بسیار انعطاف‌پذیر برای نام‌گذاری فراهم کرده‌اند به گونه‌ای که شما هر نامی می‌توانید برای متغیرتان بوسیله نویسه‌های مختلف در نوشتارهای مختلف(مثلاً چینی) انتخاب کنید. توجه داشته باشید که متغیرها در جولیا حساس به حروف هستند به این معنی که جولیا بین حروف بزرگ و کوچک تفاوت قائل می‌شود و مثلا متغیر a با A یکی نیست. برای مثال:

```
julia> x = 1.0
1.0
julia> y = -3
-3
julia> z = "My string"
"My string"
julia> CustomaryPhrase = "Hello world!"
"Hello world!"
```

برای نام‌گذاری متغیرها در جولیا می‌توانید از یونیکدهای مجاز UTF-8 استفاده کنید:

$$julia > \delta = 0.00001$$

1.0e-5

جولیا حتی به شما اجازه می‌دهد تا ثابت‌های درون‌ـساخت خود جولیا را نیز در صورت لزوم تغییر دهید. هرچند که این کار برای جلوگیری از سردرگمی اصلاً توصیه نمی‌شود:

```
julia> pi = 3
3
julia> pi
3
julia> sqrt = 4
4
```

مطلب بالا فقط زمانی صحیح است که آن ثابتی که قرار است تغییرش دهید، در حال استفاده نباشد. اگر در حال استفاده باشد، جولیا به شما خطا می‌دهد:[۴، ۱۲]

```
julia> pi
```

π = 3.1415926535897

julia> pi=3

ERROR: cannot assign variable MathConstants.pi from module Main

julia> sqrt(100)

10.0

julia> sqrt=4

ERROR: cannot assign variable Base.sqrt from module Main

۱.۲.۳ نام‌های اجازه‌داده شده برای متغیرها در جولیا

نام متغیر باید با یک حرف (A-Z) یا (a-z)، زیرخط[۸] یا یک زیر مجموعه یونیکد بزرگتر از 00A0، شروع شود؛ به طور خاص، دسته‌های نویسه یونیکد(حروف) Lu/Ll/Lt/Lm/Lo/Nl یا (ارز و سایر نمادها) Sc / So و چند نویسه letter-like (مانند یک زیر مجموعه از نمادهای ریاضی sm) مجاز هستند. نویسه‌های متعاقب بعدی نیز ممکن است شامل مجوز باشند! ارقام(۰ـ۹ و سایر نویسه‌ها در دسته Nd / No)، هم‌چنین سایر نقاط کد یونیکد: تعاریف و سایر علامت‌های اصلاح(دسته‌های Me/Mc/Sk/Mn)، برخی از نقطه‌گذاری‌های اتصال‌دهنده (دسته Pc) و چند نویسه دیگر هم مجوز دارند که در نام‌گذاری متغیر شرکت کنند. عمل‌گرها مثل "+" هم تعیین‌کننده‌های معتبری هستند اما از آن‌ها باید به صورت خاصی استفاده کرد. در برخی موارد می‌توان از عمل‌گرها(اپراتورها) به عنوان متغیر استفاده کرد. برای مثال (+) به تابع جمع اشاره دارد و f=(+) آن را دوباره تخصیص‌گذاری می‌کند. یعنی شما می‌توانید تابعی که برای جمع وجود دارد و به طور معمول عمل جمع را انجام می‌دهد، با یک تابع دیگر مثل f عوض کنید.

تنها نام های نامعتبر(اجازه داده نشده) برای متغیرها در جولیا نام‌هایی هستند که برای اسامی درون‌ـساخت به کار رفته‌اند:[۱۲]

julia> else = false

ERROR: syntax: unexpected "else"

julia> try = "No"

ERROR: syntax: unexpected "="

[8] UnderLine

۲.۲.۳ سبک‌های استاندارد نام‌گذاری جهانی

با این‌که جولیا چند محدودیت کوچک در مورد نام‌گذاری متغیرها اعمال کرده است ولی بازهم می‌توانید از قوانین جهانی نام‌گذاری استفاده کنید. این قوانین به این شرح هستند:

- متغیرها با حروف کوچک نوشته می‌شوند.
- جداسازی کلمات با استفاده از زیرخط می‌توانند نمایش داده شوند اما این کار خواندن اسامی را سخت می‌کند و فقط در جایی که خواندن کلمات با استفاده از آن آسان‌تر شود مورد استفاده قرار می‌گیرد.
- نام تایپ‌ها[۹](نوع‌ها) و نمونه‌ها[۱۰] با حروف بزرگ شروع شده و به جای استفاده از زیرخط برای جدایی کلمات از روش کوهان شتری[۱۱] استفاده می‌کنند(یعنی تمام کلمات به هم چسبیده نوشته می‌شوند و فقط حروف اول هر کلمه بزرگ نوشته می‌شود).
- نام توابع[۱۲] بدون استفاده از زیرخط و با حروف کوچک نوشته می‌شوند.
- توابعی که با نشانوندهایشان نوشته می‌شوند باید با "!" پایان یابد. گاهی این توابع را "mutating" یا "in place" می‌نامند چون که این توابع پس از فراخوانی باید در نشانوندهای خودشان تغییر ایجاد کنند نه اینکه فقط یک مقدار را برگردانند.[۵]

۳.۳ اعداد صحیح و اعشاری در جولیا

اعداد صحیح و اعشاری پایه علوم محاسباتی و محاسبات را می‌سازند. چنین مقادیری که در ساخت درونی ارائه می‌شوند را مقادیر اولیه عددی[۱۳] می‌نامند. این در حالی است که وقتی آن‌ها را بدون واسطه در برنامه‌نویسی به کار می‌گیرید، به آنها مقادیر عددی ثابت[۱۴]می‌گویند. شاید جمله بالا گویا نباشد پس مثالی می‌زنیم. برای مثال 1 در برنامه یک عدد صحیح ثابت و 1.0 یک عدد اعشاری ثابت است اما دودویی آن‌ها در حافظه به اشیائی از نوع عدد اولیه اشاره دارد.

زبان برنامه‌نویسی جولیا طیف وسیعی از انواع مقادیر اولیه عددی به همراه عملگرهای محاسباتی و بیتی را با همان استانداردهایی که توابع ریاضی‌شان تعریف می‌شوند، ارائه می‌دهد. این طرح که همگام با پشتیبانی ذاتی[۱۵]کامپیوترهای مدرن از این نوع‌ها و عملگرها است، باعث می‌شود که

[9] Type
[10] Module
[11] upper camel case
[12] Functions
[13] numeric primitives
[14] numeric literals
[15] native

جولیا بتواند از تمام امکانات محاسباتی بهره‌مند شود. علاوه بر این، جولیا یک ویژگی نرم افزاری جهت انتخاب دقت دلخواه محاسبات را ارائه می‌دهد که سخت افزارها به صورت ذاتی(بومی) آن را ندارند. اما باید توجه کنید که سرعت عمل با این کار نسبتاً کم می‌شود.[۱۲]

۱.۳.۳ اعداد صحیح

اعداد صحیح ثابت به صورت استاندارد نمایش داده می‌شوند:

```
julia> 1
1
julia> 1234
1234
```

نوع پیش فرض یک مقدار صحیح ثابت به این بستگی دارد که معماری سیستم ۳۲بیت یا ۶۴بیت باشد:

در سیستم ۳۲بیتی:

```
julia> typeof(1)
Int32
```

در سیستم ۶۴بیتی:

```
julia> typeof(1)
Int64
```

متغیر داخلی جولیا به نام Sys.WORD_SIZE نشان می‌دهد که معماری سیستم هدف، ۳۲بیتی یا ۶۴بیتی است:

در سیستم ۳۲بیتی:

```
julia> Sys.WORD_SIZE
32
```

در سیستم ۶۴بیتی:

```
julia> Sys.WORD_SIZE
64
```

جولیا هم‌چنین نوع‌های Int و UInt را تعریف کرده که نمادهایی برای اعدادصحیح علامت‌دار و بدون علامت هستند:

در سیستم ۳۲بیتی:

```
julia> Int
Int32
julia>UInt
UInt32
```

در سیستم ۶۴بیتی:

```
julia> Int
Int64
julia>UInt
UInt64
```

صرف نظر از نوع سیستم، ثابت‌های صحیح بزرگ[16] به صورت ۶۴بیت ایجاد می‌شوند: سیستم ۶۴بیت یا ۳۲بیت:

```
julia> typeof(3000000000)
Int64
```

اعداد صحیح بدون علامت با استفاده از پیشوند 0x استفاده می‌شوند. اندازه مقدار بدون علامت صحیح توسط سیستم اعداد شانزده‌تایی مشخص می‌شود:

```
julia> 0x1
0x01
julia> typeof(ans)
UInt8
julia> 0x123
0x0123
julia> typeof(ans)
UInt16
julia> 0x1234567
0x01234567
julia> typeof(ans)
UInt32
julia> 0x123456789abcdef
0x0123456789abcdef
julia> typeof(ans)
UInt64
julia> 0x111122223333444455556666777788888
0x111122223333444455556666777788888
julia> typeof(ans)
UInt128
```

این رفتار برپایه مشاهدات بنا نهاده شده است. زمانی که یک ثابت شانزده‌شانزدهی بدون علامت برای یک مقدار صحیح به کار گرفته می‌شود، به نوعی از آن‌ها برای ارائه یک رشته عددی ثابت استفاده شده نه فقط یک مقدار صحیح.

متغیر ans برای نمایش دادن آخرین نتیجه از آخرین عبارتی است که در محیط تعاملی بدست آمده. توجه کنید که ans فقط زمانی کار می‌کند که جولیا را به صورت تعاملی[17] اجرا کنید.

[16] long

[17] interactive

ثابت‌های هشت‌هشتی[18] و دو دویی[19] نیز پشتیبانی می‌شوند:
دستور برای دودویی:

```
julia> 0b10
0x02
julia> typeof(ans)
UInt8
```

دستور برای هشت‌هشتی:

```
julia> 0o010
0x08
julia> typeof(ans)
UInt8
julia>0x00000000000000001111222233334444
0x00000000000000001111222233334444
julia> typeof(ans)
UInt128
```

همان‌طور که برای ثابت شانزده‌شانزدهی گفتیم، ثابت دودویی و هشت‌هشتی هم یک نوع صحیح بدون علامت می‌سازند.
مقادیری که نمی‌توانند در UInt128 ذخیره بشوند، نمی‌توانند به عنوان ثابت‌ها نوشته شوند.
دودویی، هشت‌هشتی(اکتال) و شانزده‌شانزدهی می‌توانند به صورت علامت‌دار نیز باشند. برای اینکار باید "ـ" را بدون فاصله قبل از ثابت‌های بدون علامت بکار ببرید. آن‌ها یک عدد صحیح بی‌علامت می‌سازند با همان اندازه‌ای که یک ثابت بی‌علامت باید باشد اما با دو مقدار متمم همدیگر:

```
julia> -0x2
0xfe
```

حداقل و حداکثر مقادیر قابل نمایش برای انواع عددی اولیه به عنوان یک عدد صحیح توسط توابع typemin و typemax داده می‌شود:

```
julia> (typemin(Int32), typemax(Int32))
(-2147483648, 2147483647)
```

مقادیر بازگشتی typemin و typemax همیشه به صورت تایپ(نوع) نشانوند داده می‌شوند.[۱۲]

رفتار سرریز مقادیر

در زبان برنامه‌نویسی جولیا، گذشتن از حداکثر مقدار ارائه شده برای یک نوع موجب رفتاری موسوم به رفتار wraparound می‌شود:

```
julia> x = typemax(Int64)
```

[18] octal
[19] binary

```
9223372036854775807
julia> x + 1
-9223372036854775808
julia>x + 1 == typemin(Int64)
true
```

بنابراین محاسبات با اعداد صحیح جولیا درواقع یک فرم از مقیاس تطابق داده شده با محاسبات است. این نشان دهنده ویژگی‌های محاسباتی اعداد صحیح است که در کامپیوترهای مدرن اجرا می‌شوند. در برنامه‌هایی که امکان سرریز شدن وجود دارد، بررسی صریح برای رفتار wraparound ضروری است. اما شما می‌توانید بجای آن از نوع BigInt برای اعمال دقت مورد نظرتان استفاده کنید.[۱۲]

خطاهای تقسیم

تقسیم اعداد صحیح(تابع div) دو مورد استثنائی دارد: یکی تقسیم بر صفر و یکی تقسیم کوچکترین مقدار منفی بر 1- . هر دو مورد DivideError را نمایان می‌کنند. توابع باقی‌مانده و پیمانه(rem و mod) نیز DivideError را نمایان می‌کنند اگر نشانوند دومشان صفر باشد.[۱۲]

۲.۳.۳ اعداد اعشاری (نقطه شناور)

اعداد اعشاری ثابت در قالب استاندارد ارائه شده‌اند و در صورت نیاز می‌توانید از نماد E برای آن‌ها استفاده کنید:

```
julia> 1.0
1.0
julia> 1.
1.0
julia> 0.5
0.5
julia> .5
0.5
julia> -1.23
-1.23
julia> 1e10
1.0e10
julia> 2.5e-4
0.00025
```

تمام نتایج بالا به صورت float64 هستند. مقادیر ثابت float32 می‌توانند توسط جایگزینی f به جای e وارد شوند.

ثابت‌های شانزده‌شانزدهی اعشاری هم در دسترس هستند اما فقط به صورت float64 و حتماً باید شامل پیشوند p باشند. توجه کنید دستور به این صورت محاسبه می‌شود که: عدد قبل از p در ۲ به توان عدد بعد از p ضرب شده و حاصل نشان داده می‌شود:

```
julia> 0x1p0
1.0
julia> 0x1.8p3
12.0
julia>0x.4p-1
0.125
julia> typeof(ans)
Float64
```

برای جداسازی رقم‌ها می‌توانید از زیرخط استفاده کنید.[۱۲]

عدد اعشاری صفر

عدد اعشاری صفر، دو نوع دارد. یکی صفر مثبت و دیگری صفر منفی. هر دوی آن‌ها باهم برابرند با این تفاوت که شکل دودویی آن‌ها تفاوت دارد. شما می‌توانید این تفاوت را با استفاده از تابع bitstring مشاهده کنید:[۱۲]

```
julia> 0.0 == -0.0
true
julia> bitstring(0.0)
"0000000000000000000000000000000000000000000000000000000000000000"
julia>bitstring(-0.0)
"1000000000000000000000000000000000000000000000000000000000000000"
```

typemin و typemax برای اعداد اعشاری نیز بکار گرفته می‌شوند:

```
julia> (typemin(Float16),typemax(Float16))
(-Inf16, Inf16)
julia> (typemin(Float32),typemax(Float32))
(-Inf32, Inf32)
julia> (typemin(Float64),typemax(Float64))
(-Inf, Inf)
```

اپسیلون(مقادیر خیلی کوچک) ماشین

بسیاری از اعداد نمی‌توانند به صورت دقیق با اعداد اعشاری نمایش داده شوند و این برای بعضی از اهداف ضروری است. مثلاً نمایش فاصله دو عدد اعشاری بسیار نزدیک به هم. برای اینکه بتوانید چنین اعدادی را نمایش دهید باید از به اصطلاح اپسیلون(مقادیر بسیار کوچک) ماشین استفاده کنید.
زبان برنامه نویسی جولیا eps را مهیا کرده که فاصله بین 1.0 و بزرگترین عدد اعشاری بعد از آن را نمایش می‌دهد:

```
julia> eps(Float32)
1.1920929f-7
julia> eps(Float64)
2.220446049250313e-16
julia> eps()
2.220446049250313e-16
```

هم‌چنین تابع eps می‌تواند یک نشانوند بگیرد که نتیجه آن، قدرمطلق تفاوت میان آن نشانوند و عدد اعشاری در دسترس بعدی را نمایش می‌دهد. تابع eps(x) مقداری با همان نوع x را می‌دهد به طوری که x+eps(x) عدد اعشاری بعدی قابل دسترس و بزرگتر از x است:

```
julia> eps(1.0)
2.220446049250313e-16
julia> eps(1000.)
1.1368683772161603e-13
julia> eps(1e-27)
1.793662034335766e-43
```

مقدار فاصله بین دو عدد اعشاری مجاور، یکسان نیست و این مقدار برای اعداد کوچک‌تر، کوچک‌تر است و برای مقادیر بزرگ‌تر، بزرگ‌تر است(دلیلش این است که اعداد در کامپیوتر با یک آهنگ(rate) یکسان رشد نمی‌کنند. یعنی این طور نیست که کامپیوتر یک کوچک‌ترین عدد داشته باشد و با اضافه کردن آن به صفر(یا کم کردن آن از صفر)، بقیه اعداد را بسازد. بلکه این اعداد به وسیله معماری کامپیوتر ساخته می‌شوند). به عبارت دیگر، اعداد اعشاری حقیقی نزدیک به صفر، چگالی بیشتری دارند و هرچه از صفر دور می‌شویم، فاصله بین آن‌ها به صورت توانی زیادتر شده و تراکم آن‌ها کم‌تر می‌شود.

زبان برنامه‌نویسی جولیا nextfloat و prevfloat را نیز به ترتیب برای نمایش بزرگترین و کوچکترین عدد اعشاری موجود بعد از عدد اعشاری که به عنوان نشانوند به این توابع داده می‌شود، ارائه کرده است:

```
julia> x = 1.25f0
1.25f0
julia> nextfloat(x)
1.2500001f0
julia> prevfloat(x)
1.2499999f0
julia> bitstring(prevfloat(x))
"00111111100111111111111111111111"
julia> bitstring(x)
"00111111101000000000000000000000"
julia> bitstring(nextfloat(x))
"00111111101000000000000000000001"
```

مثالی که دیدید، این موضوع را نشان می‌دهد که دو عدد اعشاری مجاور هم، در واقع دو مقدار باینری(دو دویی) مجاور هم دارند.[۱۲، ۴]

روش گرد کردن

اگر یک عدد اعشاری به صورت دقیق(صریح) بیان نشود، باید آن را به یک مقدار مناسب گرد کنیم(تقریب بزنیم). اگر چه روش‌هایی که برای این کار استفاده می‌کنیم می‌توانند در صورت نیاز بر اساس مدل‌های موجود در IEEE 754 standard تغییر کنند(دلخواه باشند) اما روش پیش‌فرضی که در زبان برنامه‌نویسی جولیا همیشه استفاده می‌شود، RoundNearest است. این روش، به سمت نزدیک‌ترین مقدار موجود گرد می‌کند. گرد کردن به سمت نزدیکترین مقداری انجام می‌شود که کمترین تفاوت قابل توجه بیتی را برآورده می‌کند.[۱۲]

ضرایب ثابت عددی

برای همگامی با فرمول‌های مرسوم و کدهای تمیز و سرراست‌تر، در زبان برنامه‌نویسی جولیا شما می‌توانید ضرایب ثابت متغیرها را بدون واسطه و پیش از متغیرها جهت اعمال عمل ضرب میان آن‌ها به کارگیرید. با این کار بیان چند جمله‌ای‌ها بسیار صریح‌تر و تمیزتر می‌شود:[۱۲]

$$julia > x = 3$$

$$3$$

$$2x^2 - 3x + 1$$

$$10$$

$$1.5x^2 - .5x + 1$$

$$13.0$$

این کار هم‌چنین باعث ظرافت بیشتری در بیان توابع نمایی می‌شود:

$$julia > 2^2x$$

$$64$$

ثابت‌های عددی هم‌چنین به عنوان ضرایب پرانتزها نیز می‌توانند استفاده شوند:

$$julia > 2(x - 1)^2 - 3(x - 1) + 1$$

$$3$$

این نکته را فراموش نکنید که ضرایب ثابت عددی از لحاظ تقدم ریاضیاتی بر عملگرهای باینری(یعنی عملگرهایی که میان دو عملوند، عمل می‌کنند) مقدم هستند.

علاوه بر این‌ها، پرانتزها نیز می‌توانند به عنوان ضریب متغیرها جهت اعمال عمل ضرب میان آن‌ها استفاده شوند:

$$julia > (x - 1)x$$

$$6$$

شما نمی‌توانید دو پرانتز را جهت اعمال عمل ضرب در کنار یکدیگر بگذارید و هم‌چنین نمی‌توانید یک متغیر را جهت اعمال عمل ضرب قبل از یک پرانتز بگذارید:[۱۲]

$$julia > (x-1)(x+1)$$

ERROR: MethodError: objects of type Int64 are not callable

$$julia > x(x+1)$$

ERROR: MethodError: objects of type Int64 are not callable

ثابت‌های صفر و یک

زبان برنامه‌نویسی جولیا توابعی را مهیا کرده که ثابت‌های صفر و یک را باز می‌گردانند. دو نوع استفاده وجود دارد: یکی گرفتن این ثابت متناسب با یک نوع معین و یا متناسب با نوع یک متغیر:[۱۲]

تابع	توضیحات
zero(x)	گرفتن ثابت صفر از نوع x یا از متغیر x
one(x)	گرفتن ثابت یک از نوع x یا از متغیر x

مثال‌ها:

$$julia > zero(Float32)$$

0.0f0

$$julia > zero(1.0)$$

0.0

$$julia > one(Int32)$$

1

$$julia > one(BigFloat)$$

1.0

۴.۳ عملیات ریاضی و توابع اولیه

زبان برنامه‌نویسی جولیا یک مجموعه کامل از عملگرهای محاسباتی و بیتی برای کار برروی انواع نوع‌های عددی اولیه را فراهم نموده است. هم‌چنین توابع استاندارد ریاضی را به صورت همه جانبه پوشش می‌دهد.[۴]

۱.۴.۳ عملگرهای محاسباتی

عملگرهای محاسباتی زیر برای تمامی نوع‌های عددی اولیه پشتیبانی می‌شوند:[۱۲]

نماد	نام	توضیحات
+x	جمع یگانی	None
-x	تفریق یگانی	None
x+y	جمع باینری	عمل جمع را انجام می‌دهد
x-y	تفریق باینری	عمل تفریق را انجام می‌دهد
x*y	ضرب	عمل ضرب را انجام می‌دهد
x/y	تقسیم	عمل تقسیم را انجام می‌دهد
x÷y	تقسیم صحیح	قسمت صحیح x/y را برمی‌گرداند
x\y	تقسیم بالعکس	عمل y/x را انجام می‌دهد
x^y	توان	x را به توان y می رساند
y % x	باقی مانده	همان rem(x,y) است

صد البته که زبان جولیا نفی برای نوع‌های Bool را نیز پشتیبانی می‌کند:

نماد	نام	توضیحات
!x	نفی کننده	تغییر می‌دهد false را به true و بالعکس

یک سری مثال ساده را در زیر مشاهده می‌کنید:

$$julia > 1 + 2 + 3$$

6

$$julia > 1 - 2$$

-1

$$julia > 3 * 2/12$$

0.5

(بر طبق قرار داد، برای استحکام بیشتر در بیان عملگرها، ما آن‌ها را در نزدیک‌ترین حالت به عملوندشان، قبل از عملوند قرار می‌دهیم. برای مثال، ما معمولاً باید بنویسیم 2+x- تا ابتدا مقدار x را منفی کرده سپس ۲ را به آن اضافه کنیم).[۱۲]

۲.۴.۳ عملگرهای بیتی(Bitwise)

عملگرهای بیتی(Bitwise) زیر برای تمامی نوع‌های صحیح اولیه پشتیبانی می‌شوند:

نماد	نام
$\tilde{x}$	not
x&y	and
x|y	or
x> > >y	جابه‌جایی منطقی به راست
x> >y	جابه‌جایی محاسباتی به راست
x< <y	جابه‌جایی منطقی/محاسباتی به چپ

(برای نوشتن نماد xor در جولیا از veebar یا xor سپس فشردن tab می‌توانید استفاده کنید).[۱۲]
مثال‌هایی از عملگرهای بیتی در جولیا:

$$julia > 123\&234$$

$$106$$

$$julia > 123|234$$

$$251$$

$$julia > xor(123, 234)$$

$$145$$

۳.۴.۳ عملگرهای بروز رسان

عملگرهای دودویی محاسباتی و بیتی یک حالت بروز رسانی دارند که نتایج عملیات را در عملوند سمت چپ خود می‌ریزند. حالت بروز رسانی عملگرهای دودویی بوسیله استفاده بدون واسطه = پس از عملگر، به کار گرفته می‌شود. برای نمونه، نوشته می‌شود x+=3 که برابر با زمانی است که نوشته شود x=x+3 :

$$julia > x = 1$$

$$1$$

$$julia > x+ = 3$$

$$4$$

$$julia > x$$

$$4$$

تمام عملگرهای محاسباتی و بیتی حالت به روز رسانی عبارتند از:[۱۲]
+=, -=, *=, \= , /=, ÷=, %=, &=, |= , »= , «=

۴.۴.۳ مقایسات عددی(عملگرهای رابطه‌ای)

عملگرهای مقایسه‌ای(رابطه‌ای) استاندارد برای تمام نوع‌های عددی اولیه تعریف شده‌اند:[۱۲]

نام	عملگر
برابری	==
نابرابری	=!
کوچکتر از	<
کوچکتر یا مساوی	=<
بزرگتر از	>
بزرگتر یا مساوی	=>

در اینجا چند مثال ساده را مشاهده می‌کنید:

$$julia > 1 == 1$$

ture

$$julia > 5! = 5$$

false

$$julia > 1 < 2$$

true

اعداد صحیح به صورت استاندارد با مقایسه‌ی بیت‌ها مورد سنجش قرار داده می‌شوند. اعداد اعشاری نیز با توجه به استاندارد IEEE 754 مقایسه می‌شوند:[۱۲]

- اعداد محدود، به صورت معمولی مرتب شده‌اند.
- صفر مثبت برابر است با صفر منفی اما از آن بزرگتر نیست.
- Inf با خودش برابر و از هر چیز دیگری بزرگتر است بجز NaN .
- Inf با خودش برابر و از هر چیز دیگری کوچکتر است بجز NaN .
- NoN نه با چیزی برابر، نه از چیزی بزرگتر و نه کوچکتر است. شامل خودش هم می‌شود.

نکته آخر بسیار شگفت انگیز است پس آن را بهتر درک کنید:

$$julia > NaN == NaN$$

false

$$julia > NaN! = NaN$$

true

$$julia > NaN < NaN$$

false

$$julia > NaN > NaN$$

false

توابع اضافی:

تابع	تست می‌کند که آیا
isequal(x, y)	x و y یکی هستند
isfinite(x)	x محدود است
isinf(x)	x بی‌نهایت است
isnan(x)	x یک عدد نیست

isequal متغیر NaN را برابر با خودش در نظر می‌گیرد:

$$julia > isequal(NaN, NaN)$$

true

$$julia > isequal([1 NaN], [1 NaN])$$

true

$$julia > isequal(NaN, NaN32)$$

true

مقایسات زنجیره‌ای

بر خلاف بسیاری از زبان‌ها، به استثناء پایتون، مقایسه‌ها می‌توانند به طور دلخواه به صورت زنجیره‌ای استفاده شوند:

$$julia > 1 < 2 <= 2 < 3 == 3 > 2 >= 1 == 1 < 3 != 5$$

true

استفاده از مقایسات زنجیره‌ای در کدهای عددی، اغلب باعث راحتی بسیاری می‌شود. این مقایسات از عملگر && برای سنجش‌های عددی و از عملگر & برای سنجش‌های عنصری استفاده می‌کنند. این کار اجازه می‌دهد تا آن‌ها بر روی آرایه‌ها نیز عمل کنند. مثلاً:

$$julia > A = [1, 2, 3, 4, 5, 12, 24];$$

$$julia > 0. < A. < 7$$

7-element BitArray1:
true
true
true
true
true
false
false

به طرز رفتار مقایسات زنجیره‌ای توجه داشته باشید:[۱۲، ۵]

$$julia > v(x) = (println(x); x)$$

#v (generic function with 1 method)

$$julia > v(1) < v(2) <= v(3)$$

2
1
3
true

۵.۴.۳ تقدم و شرکت‌پذیری عملگرها

جولیا شرکت‌پذیری و ترتیب‌دهی عملگرها را به صورت زیر اعمال می‌کند. ترتیب جدول زیر از بیشترین تقدم به کمترین تقدم است:[۱۲، ۵]

شرکت‌پذیری	عملگرها	دسته
چپ	.	نحو
راست	^	توان
راست	+ - √	یگانی
چپ	« »	بیت جابه‌جایی‌ها
چپ	//	کسری‌ها
چپ	* / % & \ ÷	ضرب
چپ	+ - |	جمع
چپ	: . ..	نحو
چپ	<|	نحو
راست	|<	نحو
بدون شرکت‌پذیری	< > >= <= == === != =! ==: :>	مقایسات
راست	&&	کنترل روند
راست	<=	جفت کردن
راست	= %= |= &= ÷= \= ^= //= /= *= -= +=	تخصیص‌ها

شما هم‌چنین می‌توانید تقدم به صورت عددی برای هر عملگر داده شده را از طریق تابع پیش ساخته[۲۰] پیدا کنید؛ اعداد بالاتر دارای تقدم بالاتری هستند:[۱۲]

$$julia > Baseoperator\_precedence(:+), Baseoperator\_precedence(:*), Baseoperator\_precedence(:$$

(11, 13, 17)

۶.۴.۳ تبدیل‌های عددی

زبان برنامه‌نویسی جولیا از تبدیلات عددی به سه شکل پشتیبانی می‌کند که تفاوت این شکل‌ها در صحت دقت تبدیلات‌شان است.

- نشانه‌گذاری T(x) یا convert(T,x) درواقع x را به مقداری با نوع T تبدیل می‌کند.
- x%T یک عدد صحیح x را به یک مقدار با نوع صحیح T تبدیل می‌کند به صورتی که x به پیمانه دو به توان n باشد و در آن n تعداد بیت‌ها در T است. به عبارت دیگر، ارائه دودویی کوتاه شده است تا سازگاری ایجاد شود.
- تابع گرد کردن، می‌تواند یک نوع T را به صورت اختیاری به عنوان یک نشانوند بگیرد. مثلاً، round(Int,x) کوتاه شده عبارت Int(round(x)) است.

[20] Base.operator_precedence

مثال زیر تفاوت این سه شکل را روشن می‌سازد:[۱۲، ۵]

$$julia > Int8(127)$$
127

$$julia > Int8(128)$$
ERROR: InexactError: trunc(Int8, 128)

$$julia > Int8(127.0)$$
127

$$julia > Int8(3.14)$$
ERROR: InexactError: Int8(3.14)

$$julia > Int8(128.0)$$
ERROR: InexactError: Int8(128.0)

$$julia > 127\%Int8$$
127

$$julia > 128\%Int8$$
-128

$$julia > round(Int8, 127.4)$$
127

$$julia > round(Int8, 127.6)$$
ERROR: InexactError: trunc(Int8, 128.0)

تابع	توضیح	تایپ بازگشتی
round(x)	x را به نزدیک‌ترین عدد صحیح گرد می‌کند	typeof(x)
round(T,x)	x را به نزدیک‌ترین عدد صحیح گرد می‌کند	T
floor(x)	x را به سمت Inf- گرد می‌کند	typeof(x)
floor(T,x)	x را به سمت Inf- گرد می‌کند	T
ceil(x)	x را به سمت Inf+ گرد می‌کند	typeof(x)
ceil(T,x)	x را به سمت Inf+ گرد می‌کند	T
trunc(x)	x را به سمت صفر گرد می‌کند	typeof(x)
trunc(T,x)	x را به سمت صفر گرد می‌کند	T

جدول ۱.۳: توابع گرد کننده

تابع	توضیح
div(x,y)	تقسیم truncated
fld(x,y)	تقسیم زمینی floored
cld(x,y)	تقسیم ceiling
rem(x,y)	باقی مانده
mod(x,y)	حساب پیمانه‌ای
mod1(x,y)	mod با وزن متعادل کننده ۱
mod2pi(x)	حساب به پیمانه 2pi
divrem(x,y)	(div(x,y),rem(x,y)) را باز می‌گرداند
fldmod(x,y)	(fld(x,y),mod(x,y)) را باز می‌گرداند
gcd(x,y)	بزرگترین مقسوم علیه مشترک(ب م م) x ، y
lcm(x,y)	کوچکترین مضرب مشترک(ک م م) x ، y

جدول ۲.۳: توابع تقسیم

تابع	توضیح
abs(x)	یک مقدار مثبت با اندازه‌ی x
abs2(x)	توان دوم اندازه‌ی x
sign(x)	علامت x را نشان می‌دهد
signbit(x)	نشان می‌دهد که آیا بیتِ علامت روشن یا خاموش است
copysign(x,y)	یک مقدار با اندازه x و با علامت y را برمی‌گرداند
flipsign(x,y)	یک مقدار با اندازه x و با علامت x*y را برمی‌گرداند

جدول ۳.۳: توابع علامت و قدرمطلق

تابع	توضیح
sqrt(x)	ریشه‌ی دوم x
cbrt(x)	ریشه‌ی سوم x
hypot(x,y)	وتر مثلث قائم الزاویه‌ای را می‌دهد که اضلاع دیگر آن y و x هستند
exp(x)	تابع e^x
expm1(x)	مقدار دقیق exp(x)-1 برای x نزدیک به صفر
ldexp(x,n)	x*2^n را برای n های صحیح محاسبه می‌کند
log(x)	لگاریتم طبیعی(بر مبنای عدد نپر) را برای x می‌دهد
log(b,x)	لگاریتم x بر مبنای b
log2(x)	لگاریتم x بر مبنای ۲
log10(x)	لگاریتم x بر مبنای ۱۰
log1p(x)	به دقت log(1+x) را برای هایx نزدیک به صفر حساب می‌کند
exponent(x)	binary exponent of x
significand(x)	binary significand (a.k.a. mantissa) of a floating-point number x

جدول ۴.۳: توان‌ها، لگاریتم‌ها و ریشه‌ها

توابع مثلثاتی و هایپربولیک

تمام توابع استاندارد مثلثاتی و هایپربولیک نیز تعریف شده‌اند:

sin cos tan cot sec csc sinh cosh tanh coth sech csch asin acos atan acot asec acsc asinh acosh atanh acoth asech acsch sinc cosc

تمامی این توابع، تک نشانوندی هستند در حالی که atan می‌تواند دو نشانوند را در تابع atan2 بپذیرد.[۱۲]

۵.۳ اعداد گویا و مختلط در جولیا

زبان برنامه‌نویسی جولیا نوع‌های از پیش تعریف شده‌ای برای هر دوی اعداد گویا و مختلط را در خود دارد و تمام عملیات ریاضیاتی و توابع اولیه مربوط به آن‌ها را پشتیبانی می‌کند. تبدیل و ارتقاء نیز تعریف شده‌اند که در نتیجه عملیات روی هر ترکیب از نوع‌های عددی از پیش تعریف شده، خواه نوع‌های اولیه و خواه ترکیب شده باشند، همان‌طور که انتظار می‌رود رفتار می‌کند.[۵]

۱.۵.۳ اعداد مختلط

ثابت جهانی im برای عدد مختلط i (که ریشه دوم عدد ۱ـ است) تخصیص داده شده است. از آن‌جایی که زبان برنامه‌نویسی جولیا اجازه می‌دهد که ثابت‌های عددی به عنوان ضریب متغیرها بدون فاصله قبل از آن‌ها قرار گیرند، این قرارگیری به اندازه کافی برای ارائه نحو مناسب برای اعداد مختلط، شبیه به نماد سنتی ریاضی است:[۱۲، ۵]

$$julia > 1 + 2im$$

$$1 + 2\text{im}$$

شما می‌توانید تمام عملیات ریاضی استاندارد را بر روی اعداد مختلط انجام دهید:

$$julia > (1 + 2im)/(1 - 2im)$$

$$-0.6 + 0.8\text{im}$$

$$julia > (1 + 2im) + (1 - 2im)$$

$$2 + 0\text{im}$$

$$julia > (-3 + 2im) - (5 - 1im)$$

-8 + 3im

$$julia > (-1 + 2im)\hat{}2$$

-3 - 4im

مکانیسم ارتقاء، اطمینان می‌دهد که ترکیب‌های عملوندهای مختلف دارای نوع‌های متفاوت، به درستی کار کند:

$$julia > 2(1 - 1im)$$

2 - 2im

$$julia > 0.75(1 + 2im)$$

0.75 + 1.5im

$$julia > 1 + 3/4im$$

1.0 - 0.75im

نکته

توجه داشته باشید که 3/4im == 3/(4*im) == -(3/4*im) و این به آن دلیل است که یک ضریب ثابت چسبیده به متغیر، تقدم بیشتری نسبت به تقسیم دارد.

توابع استانداردی جهت مدیریت مقادیر مختلط تدارک دیده شده است:

$$julia > z = 1 + 2im$$

1 + 2im

$$julia > real(1 + 2im)\#realpartofz$$

1

$$julia > imag(1 + 2im)\#imaginarypartofz$$

2

$$julia > conj(1 + 2im)\#complexconjugateofz$$

1 - 2im

$$julia > abs(1 + 2im)\#absolutevalueofz$$

2.23606797749979

$$julia > abs2(1 + 2im)\#squaredabsolutevalue$$

5

$$julia > angle(1 + 2im)\#phaseangleinradians$$

1.1071487177940904

محدوده وسیع دیگر توابع ابتدایی، برای اعداد مختلط نیز تعریف شده‌اند:

$$julia > sqrt(1im)$$

0.7071067811865476 + 0.7071067811865475im

$$julia > cos(1 + 2im)$$

2.0327230070196656 - 3.0518977991518im

نماد گذاری ضریب عددی ثابت برای ساخت عدد مختلط، کارایی ندارد. به جای آن، ضرب باید به صورت صریح بیان شود:

$$julia > a = 1; b = 2; a + b * im$$

$$1 + 2\text{im}$$

با این حال، این کار توصیه نمی‌شود. به جای آن از تابع مؤثر complex برای ساخت مقدار مختلط استفاده کرده و آن را به طور مستقیم از قسمت‌های حقیقی و موهومی‌اش پدید آورید:

$$julia > a = 1; b = 2; complex(a, b)$$

$$1 + 2\text{im}$$

این نوع ساخت، از ضرب و عملیات اضافی جلوگیری می‌کند.[۱۲]

۲.۵.۳ اعداد گویا

زبان برنامه‌نویسی جولیا دارای نوع عدد گویا است که نسبت‌های دقیق اعداد صحیح را نمایش می‌دهد. اعداد گویا با استفاده از عملگر // ساخته می‌شوند:

$$julia > 2//3$$

$$2//3$$

اگر صورت و مخرج یک نسبت دارای ضریب‌های مشترک باشند، با یک دیگر ساده شده و به شکل کمترین مقدار غیر قابل ساده شدن در می‌آیند و البته مخرج به صورت غیر منفی است:

$$julia > 6//9$$

$$2//3$$

این شکل نرمال شده برای یک نسبت صحیح به صورت منحصر به فرد است، بنابراین برابری مقادیر گویا را می‌توان به وسیله چک کردن برابری صورت و مخرج آن‌ها تست کرد. بوسیله توابع numerator و denominator می‌توان صورت و مخرج استاندارد شده‌ی یک مقدار گویا را از هم جدا کرد:

$$julia > numerator(2//3)$$

2

$$julia > denominator(2//3)$$

3

زمانی که عملگرهای استاندارد محاسبه و مقایسه برای مقادیر گویا نیز تعریف شده‌اند، به مقایسه مستقیم صورت و مخرج واقعاً نیازی نیست:

$$julia > 2//3 == 6//9$$

true

اعداد گویا به شیوه بسیار راحتی می‌توانند به مقادیر اعشاری تبدیل شوند:

$$julia > float(3//4)$$

0.75

تبدیل از گویا به اعشاری، بر طبق هویتِ(نوع‌هایی که صورت و مخرج دارند) مقادیرِ بدون کسرِ a و b انجام می‌شود. البته به غیر از موردی که a==0 و b==0 .

$$julia > a = 1; b = 2;$$

$$julia > isequal(float(a//b), a/b)$$

true

ساختار مقدار گویای بی‌نهایت نیز قابل قبول است:

$$julia > 5//0$$

$$julia > -3//0$$

$$julia > typeof(ans)$$

RationalInt64

هر چقدر هم سعی کنید که یک مقدار گویای NaN بسازید، با این حال، این مقدار بی‌اعتبار است:

$$julia > 0//0$$

ERROR: ArgumentError: invalid rational: zero(Int64)//zero(Int64)

به طور معمول، بوسیله سیستم ارتقاء زبان برنامه‌نویسی جولیا، تعاملات با دیگر نوع‌های عددی آسان شده است:[۱۲]

$$julia > 3//5 + 1$$

$$8//5$$

$$julia > 2//7 * (1 + 2im)$$

$$2//7 + 4//7*\text{im}$$

۶.۳ رشته‌ها در جولیا

رشته‌ها[۲۱]دنباله‌هایی محدود از نویسه‌ها هستند. البته مشکل اصلی وقتی پیش می‌آید که یک نفر بپرسد: "نویسه چیست؟". نویسه‌هایی که انگلیسی زبانان با آن آشنا هستند عبارتند از حروف A و B و C و ... به همراه ارقام و علامت‌های نشانه‌گذاری مشترک. این نویسه‌ها به همراه یک نگاشت به مقادیر صحیح میان 0 و 127 بوسیله استاندارد ASCII، استانداردسازی شده‌اند.

صد البته که نویسه‌های بسیار دیگری خارج از زبان انگلیسی نیز استفاده شده‌اند از جمله انواع دیگری از نویسه‌های ASCII با لهجه‌ها و اصلاحات دیگر، مانند فایل آغازگرهای سریلانکایی و یونانی و فایل آغازگرهایی که کاملاً نامربوط به ASCII و انگلیسی هستند، شامل: عربی، چینی، عبری، هندی، ژاپنی و کره‌ای.

ابزار استاندارد یونیکد، پیچیدگی این‌که نویسه دقیقاً چیست را حل کرده و عموماً به عنوان استاندارد قطعی حل این مسئله شناخته می‌شود. با توجه به نیازتان، می‌توانید این پیچیدگی‌ها را نادیده بگیرید و وانمود کنید که فقط نویسه‌های ASCII وجود دارند. اما می‌توانید کدهای خود را بوسیله نویسه‌ها یا رمزگذاری‌هایی ایجاد کنید که ASCII نیستند.

[21] String

جولیا همان‌قدر که با متن‌های ASCII ساده و مؤثر برخورد می‌کند، همان‌قدر هم با یونیکد رفتار می‌کند. به طور خاص، شما می‌توانید کدهای رشته‌ی خود را در استایل C بنویسید برای پردازش رشته‌های ASCII. این کار همان‌طوری که انتظار دارید کار می‌کند، هم از لحاظ کارایی و هم از لحاظ معنایی.

اگر کدی با یک متن غیر ASCII برخورد کند، یک پیام خطای واضح را به جای یک نتیجه غیر قابل فهم نمایش می‌دهد تا شما بتوانید به راحتی اشتباهات مربوط به داده‌های غیر ASCII را درک کنید.

در این جا یک سری ویژگی‌های قابل توجهی درباره رشته‌های جولیا وجود دارد:
رشته‌ها با " تعریف می‌شوند:

"This is a string"

نویسه‌ها با ' مشخص می‌شوند:

'a'

رشته‌ها مانند آرایه‌ای از نویسه‌ها قابل اندیس‌گذاری می‌باشند:

"This is a string"[1] # => 'T' # Julia indexes from 1

البته این روش برای رشته‌های با کدگذاری UTF8 مناسب نیست.
نویسه $ عبارات جولیا را در داخل یک رشته بیان می‌کند:

str = "2 + 2 = $(2 + 2)" # => "2 + 2 = 4"

روش جایگزین جهت تعریف قالب‌های چاپی برای رشته‌ها استفاده از تابع[۲۲] printf می‌باشد:

@printf "%d is less than %f" 4.5 5.3 # 5 is less than 5.300000

تولید خروجی قابل چاپ ساده می‌باشد:

println("I'm Julia. Nice to meet you!")

تبدیل نویسه به عدد:

$$julia > Int('x')$$

120

[22] macro

$$julia > typeof(ans)$$

Int64

تبدیل عدد به نویسه:

$$julia > Char(120)$$

'x'

برای مقایسه داریم:

$$julia >' A' <' a'$$

true

$$julia >' A' <=' a' <=' Z'$$

false

برای استخراج یک نویسه از رشته داریم:

$$julia > str = "Hello, world."$$

"Hello, world."

$$julia > str[6]$$

','

برای استخراج زیررشته از رشته داریم:

$$julia > str[4:9]$$

"lo, wo"

همچنین می‌توان از تابع SubString نیز استفاده کرد:

$$julia > str = "longstring"$$

"long string"

$$julia > substr = SubString(str, 1, 4)$$

"long"

$$julia > typeof(substr)$$

SubString{String}

یکی از رایج‌ترین موارد مورد استفاده از رشته‌ها عمل چسباندن[۲۳] است:

$$julia > greet = "Hello"$$

"Hello"

$$julia > whom = "world"$$

"world"

$$julia > string(greet, ", ", whom, ".")$$

"Hello, world."

با استفاده از نماد * نیز می‌توان عمل چسباندن را انجام داد:

$$julia > greet * ", " * whom * "."$$

"Hello, world."

[23] Concatenation

برای پیدا کردن شاخص[۲۴]یک حرف در رشته داریم:

$$julia > findfirst(isequal('o'), "xylophone")$$

4

$$julia > findlast(isequal('o'), "xylophone")$$

7

تابع occursin چک می‌کند که یک زیررشته در رشته موجود هست یا نه:

$$julia > occursin("world", "Hello, world.")$$

true

توابع تکرار[۲۵]و پیوستن[۲۶]:[۱۲، ۵]

$$julia > repeat(". : Z : .", 10)$$

".:Z:..:Z:..:Z:..:Z:..:Z:..:Z:..:Z:..:Z:..:Z:..:Z:."

$$julia > join(["apples", "bananas", "pineapples"], ", ", ", and")$$

"apples, bananas and pineapples"

۷.۳ توابع در جولیا

شکل اصلی تعریف تابع در جولیا به شکل زیر است:

$$julia > f(x, y) = x + y$$

$$julia > g = f;$$

$$julia > g(2, 3)$$

5

[24] index
[25] repeat
[26] join

شکل دیگر برای تعریف:

```
julia> function hypot(x,y)
           x = abs(x)
           y = abs(y)
           if x > y
               r = y/x
           return x*sqrt(1+r*r)
           end

           if y == 0
           return zero(x)

           end

           r = x/y
           return y*sqrt(1+r*r)

           end
```

$$julia > hypot(3, 4)$$

5.0

با استفاده از اپراتور"::" می‌توان نوع[۲۷] برگشتی را مشخص کرد:[۱۲]

```
julia> function g(x, y)::Int8
           return x * y
           end;
julia> typeof(g(1, 2))
Int8
```

۸.۳ روند کنترل

روند کنترل[۲۸] به ۴ صورت است:[۱۲، ۴]

1.Compound Expressions: begin and (;).
2.Conditional Evaluation: if-elseif-else and ?: (ternary operator).
3.Short-Circuit Evaluation: &&, || and chained comparisons.
4.Repeated Evaluation: Loops: while and for.

[27] type
[28] control flow

۱.۸.۳ عبارات مرکب

نمونه برای کاربرد begin:

```
julia> z = begin
x = 1
y = 2
x + y
end
3
```

نمونه برای کاربرد "(;)" :[۱۲]

```
julia> z = (x = 1; y = 2; x + y)
3
```

۲.۸.۳ عبارات شرطی

کاربرد if-elseif-else :

```
if x < y
println("x is less than y")

elseif x > y
println("x is greater than y")

else
println("x is equal to y")

end;
```

کاربرد"?:" :[۱۲]

$$julia > x = 1; y = 2;$$

$$julia > println(x < y?"lessthan" : "notlessthan")$$

less than

$$julia > x = 1; y = 0;$$

$$julia > println(x < y?"lessthan" : "notlessthan")$$

not less than

۳.۸.۳ ارزیابی اتصالات کوتاه

اولویت && از || بیشتر است. مثال زیر را ببینید:[۱۲، ۵]

$$julia > t(x) = (println(x); true)$$

$$julia > f(x) = (println(x); false)$$

$$julia > t(1)\&\&t(2)$$

1
2
true

$$julia > t(1)\&\&f(2)$$

1
2
false

$$julia > t(1)||t(2)$$

1
true

۴.۸.۳ حلقه‌ها

دو ساختار برای حلقه‌ها وجود دارد:[۱۲]

1.while

$$julia > i = 1;$$

```
julia> while i <= 5
    println(i)
    global i += 1
    end
```

```
1
2
3
4
5
```

1.for

```
julia> for i = 1:5
    println(i)
    end
```

```
1
2
3
4
5
```

۵.۸.۳ سازنده‌ها

سازنده‌ها[۲۹] یک روش خاص است که برای مقداردهی اولیه‌ی میدان‌های کلاس و پرکردن شیء[۳۰] با داده به کار می‌رود. تابع constructor در زمان ایجاد شیء فراخوانی می‌شود. این تابع در واقع مقادیر را ساخته، میدان‌های کلاس را مقداردهی می‌کند و شیء ساخته شده از روی کلاس را با داده پر می‌نماید.

مثال زیر را از سازنده‌ها در زبان جولیا ببینید:[۱۲، ۵]

[29] constructors

[30] Object

```
julia> struct Foo
           bar
           baz
       end

julia> foo = Foo(1, 2)
Foo(1, 2)

julia> foo.bar
1

julia> foo.baz
2
```

فصل ۴

نتیجه گیری

این پژوهش به منظور«آشنایی با زبان جولیا، نشان دادن اهمیت این زبان، کاربردهایش، مقایسه‌اش با دیگر زبان‌های پرکاربرد و آموزش مختصر این زبان» انجام شد.

با استفاده از یافته‌های فصل اول مشخص شد که زبان جولیا دارای ویژگی‌های بسیار کارآمدی می‌باشد. ویژگی‌هایی مانند فراخوانی توابع C و پایتون، طراحی شده برای موازی سازی که ویژگی بسیار ارزشمندی است، قابلیت ارسال چندگانه و سرعت بالا، نحو ریاضی بسیار عالی، نوع اختیاری و کلی کاربردهای منحصر به فرد و برجسته‌ی دیگر.

با استفاده از یافته‌های فصل دوم مشخص شد که جولیا به دنبال چه اهدافی است. جولیا به دنبال افزایش سرعت محاسبات، ارائه ترکیب نحوی ساده و استفاده از مزایای نوع‌های ایستا و پویا به یک اندازه است. جولیا را با زبان پیشرفته‌ای مثل پایتون مقایسه کردیم و دریافتیم که ویژگی‌های مثبت جولیا حتی از پایتون هم بیشتر است. سرعت جولیا از پایتون بهتر است، ترکیب نحوی متناسب با ریاضیات در زبان جولیا ایده‌آل است، مانند پایتون کاربر را درگیر جزییات مربوط به تخصیص و آزادسازی حافظه نمی‌کند و هم‌چنین انجام عملیات به شکل موازی است. در کل سرعت اجرا توابع منتخب در جولیا بسیار بالاتر از پایتون است. حدس من این است که جولیا به زودی جایگزین پایتون خواهد شد و در حال حاضر بهترین جایگزین نیز می‌باشد.

واژه‌نامه فارسی به انگلیسی

جولیا ..... Julia
پایتون ..... Python
آر ..... R
کامپایلر ..... Compiler
سی C
فورترن ..... Fortran
ویرال شاه ..... Viral Shah
جف بزانسون ..... Jeff Bezanson
استفان کارپینسکی ..... Stefan Karpinski
محاسبات موازی ..... Parallel Computing
آلن ادلمن ..... Alan Edelman
بارگیری ..... Download
ارسال چندگانه ..... Multiple Sending
سیستم پویا ..... Dynamic System
نوع ..... Type
حساب گوگل ..... Google Accont
بسته ..... Packet
محیط توسعه مجتمع ..... Integrated Development Environment(IDE)
خط دستور ..... Command Line
زیست بوم ..... Eco System
یادگیری ماشین ..... Machine Learning
تابع ..... Macro
لیسپ ..... Lisp Language
متلب ..... Matlab language
روبی ..... Ruby Language
مدیریت خودکار حافظه ..... Garbage Collection
تعویض ..... Switch

بسته بندی ..... Package
برنامه‌نویسی پویا ..... Meta-Programming
رویه ..... Method
جامع ..... Generic
سند ..... Script
پرل ..... Perl
حساس به حروف ..... Case-sensetive
نمونه ..... Module
ثابت‌های صحیح بزرگ ..... Long
تعاملی ..... Interactive
ثابت‌های هشت‌هشتی ..... Octal
ثابت‌های دودویی ..... Binary
مقادیر اولیه عددی ..... Numeric Primitives
مقادیر عددی ثابت ..... Numeric Literals
ذاتی ..... Native
شانزده‌شانزدهی ..... Hex
مقدار بسیار کوچک ..... Epsilon
گرد کردن ..... Rond
روش ..... Method
جابه‌جایی ..... Shift
تابع پیش ساخته ..... Base Operator Precedence
روند کنترل ..... Control Flow
سازنده ..... Contructor
میدان ..... Field
شیء ..... Object

کتاب‌نامه

[1] Stephen Boyd and Lieven Vandenberghe. Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares.

[2] Mykel J. Kochenderfer, Tim A. Wheeler. Algorithms for Optimization (The MIT Press).

[3] Julia Programming for Operations Research.

[4] Allen Downey, Ben Lauwens. Think Julia.

[5] Paul D. McNicholas, Peter Tait. Data Science with Julia.

[6] Ivo Balbaert. Getting Started with Julia.

[7] https://en.wikipedia.org/wiki/Julia_(programming_language). .

[8] https://julialang.org/downloads/. .

[9] https://docs.julialang.org/en/v0.6.2/manual/packages/. .

[10] https://julialang.org/ecosystems/. .

[11] https://julialang.org/. .

[12] https://docs.julialang.org/en/v1/manual/. .

Abstract

Scientific computing has traditionally required the highest performance, yet domain experts have largely moved to slower dynamic languages for daily work. We believe there are many good reasons to prefer dynamic languages for these applications, and we do not expect their use to diminish. Fortunately, modern language design and compiler techniques make it possible to mostly eliminate the performance trade-off and provide a single environment productive enough for prototyping and efficient enough for deploying performance-intensive applications. The Julia programming language fills this role: it is a flexible dynamic language, appropriate for scientific and numerical computing, with performance comparable to traditional statically-typed languages.

Because Julia's compiler is different from the interpreters used for languages like Python or R, you may find that Julia's performance is unintuitive at first. If you find that something is slow, we highly recommend reading through the Performance Tips section before trying anything else. Once you understand how Julia works, it's easy to write code that's nearly as fast as C.

Julia features optional typing, multiple dispatch, and good performance, achieved using type inference and just-in-time (JIT) compilation, implemented using LLVM. It is multi-paradigm, combining features of imperative, functional, and object-oriented programming. Julia provides ease and expressiveness for high-level numerical computing, in the same way as languages such as R, MATLAB, and Python, but also supports general programming. To achieve this, Julia builds upon the lineage of mathematical programming languages, but also borrows much from popular dynamic languages, including Lisp, Perl, Python, Lua, and Ruby.

The most significant departures of Julia from typical dynamic languages are:

The core language imposes very little; Julia Base and the standard library is written in Julia itself, including primitive operations like integer arithmetic A rich language of types for constructing and describing objects, that can also optionally be used to make type declarations The ability to define function behavior across many combinations of argument types via multiple dispatch Automatic generation of efficient, specialized

code for different argument types Good performance, approaching that of statically-compiled languages like C Although one sometimes speaks of dynamic languages as being "typeless", they are definitely not: every object, whether primitive or user-defined, has a type. The lack of type declarations in most dynamic languages, however, means that one cannot instruct the compiler about the types of values, and often cannot explicitly talk about types at all. In static languages, on the other hand, while one can – and usually must – annotate types for the compiler, types exist only at compile time and cannot be manipulated or expressed at run time. In Julia, types are themselves run-time objects, and can also be used to convey information to the compiler.

While the casual programmer need not explicitly use types or multiple dispatch, they are the core unifying features of Julia: functions are defined on different combinations of argument types, and applied by dispatching to the most specific matching definition. This model is a good fit for mathematical programming, where it is unnatural for the first argument to "own" an operation as in traditional object-oriented dispatch. Operators are just functions with special notation – to extend addition to new user-defined data types, you define new methods for the + function. Existing code then seamlessly applies to the new data types.

Partly because of run-time type inference (augmented by optional type annotations), and partly because of a strong focus on performance from the inception of the project, Julia's computational efficiency exceeds that of other dynamic languages, and even rivals that of statically-compiled languages. For large scale numerical problems, speed always has been, continues to be, and probably always will be crucial: the amount of data being processed has easily kept pace with Moore's Law over the past decades.

Julia aims to create an unprecedented combination of ease-of-use, power, and efficiency in a single language

College of Science
School of Mathematics, Statistics, and Computer Science

Introducing and learning Julia's language

Auther: Sarina Sefidgar hosseini

Advisor: Engineer ebrahim naghibzade mashayekh

A thesis submitted to Graduate Studies Office
in partial fulfillment of the requirements for the degree of
B.Sc. Computer Science