1_variable_easycalc

2017 年 3 月 16 日

1 Introduction to Data Science with Julia

1.1 目次

- Hello-World!
- コメント
- Help
- 簡単な数値計算
- 変数
- 比較演算子
- 論理演算
- 文字列
- 練習問題

2 Hello World!

まずはプログラミング言語を学ぶ時の定番の Hello World! 出力するコードを書いてみましょう。

Julia で文字を表示するには print または println 関数を使います。 print と println の違いは最後に改行コードが入るか否かの違いです。

```
print("hoge\n"), println("hoge")
```

は同じ意味です。

実際に、次の空欄に

```
print("Hello World!")
```

と入力し、Shift-Enter または Ctrl-Enter で実行してみましょう。

In [ ]:

目次に戻る

3 コメント

プログラミングが上手くなるためにはひたすらコードを書いて慣れるしかありません。しかし、コードを書き連ねていると後で見なおした時に一体何の計算をしていたのかわからなくなってしまいことが多々起きま

す。未来の自分が見ても理解できるように、プログラミングをするときにはどんな計算をしているのかというコメントを入れることを意識しましょう。

Julia ではコメントにしたい部分は # のあとに書きます。# の後に書いたものは例えコンピュータが実行できる文法だったとしても評価されません。

```
In [1]: # Hello world を出力するプログラム。
        print("Hello World!")
        # print("Kyushu University")
```

```
Hello World!
```

複数行に渡る長いコメントを書く場合は #= =# で囲みます。

```
In [2]: #=
        ここに書いたことは評価されません。
        プログラムを書き直すときにコードの不必要な部分を消してしまう人がいますが、後々になって消さなければよかっ
        そのため、一度は不必要だと思っても消さずに、このようにコメントとして残しておくことをおすすめします。
        =#
```

目次に戻る

4 Help

プログラミンの上達のためには、自ら手を動かしてコードを書くことも重要ですが、プログラミングが上手い人のコードを真似ることも重要です。ですが、他人のコードを参考にしていると必ずあなたの知らない関数と出会うことでしょう。

Julia では、そのような未知の関数の動作を知りたい場合は

?関数名

とすることで使い方を調べることが出来ます。ただし、全ての関数に Help があるわけではないので、そのようなときはネットで調べるか、公式ドキュメントを当たってください。

```
In [3]: ?fft
```

```
search: fft fft! FFTW fftshift rfft ifft bfft ifft! bfft! ifftshift irfft brfft
```

```
Out[3]:
```

```
fft(A [, dims])
```

Performs a multidimensional FFT of the array `A`. The optional `dims` argument specifies an iterable subset of dimensions (e.g. an integer, range, tuple, or array) to transform along. Most efficient if the size of `A` along the transformed dimensions is a product of small primes; see `nextprod()`. See also `plan_fft()` for even greater efficiency.

A one-dimensional FFT computes the one-dimensional discrete Fourier transform (DFT) as defined by

$$\text{DFT}(A)[k] = \sum_{n=1}^{\text{length}(A)} \exp\left(-i\frac{2\pi(n-1)(k-1)}{\text{length}(A)}\right) A[n].$$

A multidimensional FFT simply performs this operation along each transformed dimension of `A`.

!!! note * Julia starts FFTW up with 1 thread by default. Higher performance is usually possible by increasing number of threads. Use `FFTW.set_num_threads(Sys.CPU_CORES)` to use as many threads as cores on your system. * This performs a multidimensional FFT by default. FFT libraries in other languages such as Python and Octave perform a one-dimensional FFT along the first non-singleton dimension of the array. This is worth noting while performing comparisons. For more details, refer to the "Noteworthy Differences from other Languages" section of the manual.

```
In [ ]:
```

目次に戻る

4.1 簡単な数値計算

千里の道も一歩から。まずは簡単な四則演算からプログラミングの初歩を学びましょう。次の空欄に 1 + 1 や 2 * 8 など簡単な計算式を入力し、Shift - Enter で実行してみましょう。

他のプログラミング言語同様、Julia での四則演算は +, -, *, / を使います。

```
In [4]: 1 + 3
```

```
Out[4]: 4
```

プログラミング言語によっては

整数 / 整数 = 整数

となりますが、Julia では小数になります。

```
In [5]: 3 / 2 # C言語で同様のことをすると 1 になります。
```

```
Out[5]: 1.5
```

商を求めるには div 関数を、余りを求めるには mod, rem, % のいずれかを使います。

正の整数同士の剰余算では mod, rem も同じ値を返しますが、負の整数と正の整数では mod と rem で違う値を返します。気になる人は ?mod, ?rem でどのように違うのか調べてみましょう。

```
In [6]: div(5, 2)
```

```
Out[6]: 2
```

```
In [7]: mod(11, 3)
```

```
Out[7]: 2
```

```
In [8]: 10 % 3
```

```
Out[8]: 1
```

divrem 関数を使うと商と余りを一度に計算することが出来ます。

```
In [9]: divrem(5,2)

Out[9]: (2,1)

In [ ]:
```

目次に戻る

他の一般的なプログラミング言語と違い Julia では分数も扱うことが出来ます。分数同士の四則演算では結果も分数になりますが、分数 * 小数 などとすると結果は小数になります。$\frac{a}{b}$ を表現するためには

```
a // b
```

と打ちます。

```
In [10]: 3//2

Out[10]: 3//2

In [11]: 2//6 # 自動的に約分される

Out[11]: 1//3

In [12]: 2//3 + 3//4

Out[12]: 17//12

In [13]: 2//3 * 6

Out[13]: 4//1

In [14]: 2//3 * 1.5

Out[14]: 1.0
```

その他よく使うものとして、べき乗は ^ を使います。

```
In [15]: 2^6 # 2の6乗

Out[15]: 64

In [ ]:
```

目次に戻る

4.2 変数

電卓代わりに使うのならば上記のような使い方でも良いですが、プログラムでは一度計算した結果を再度使用することが多々有ります。そのたびに計算式を書いていては非効率的です。このようなときは変数を用いて計算結果を保存しましょう。変数 x に 10 を保存する場合は以下のようにします。

```
x = 10
```

数学では = は等号ですが、プログラミングの世界では代入を表します。つまり、x = 10 と書いたら、 x という名前がついた箱に 10 という数字を格納することを意味します。

C 言語などのコンパイラ型言語では

```
int x = 10
```

と変数の型を宣言する必要がありますが、Julia ではそのような型宣言は必要ありません。

変数名にはローマ字以外にもアンダーバー" _ " などの記号も使えますが、以下のものは使えません。 - ピリオド . コンマ , を入れた変数名 - for, if などのすでに特殊な意味を持つ文字 - 数字から始まるもの

Good names - hoge - Hoge # Julia では大文字と小文字は区別されます - piyo_foo - _tux - 九州 # Julia では漢字も変数名として使えます。

Bad names - hoge.piyo - 123foo - if

変数名には文法上許す限りどんなものでも使えますが、Julia 推奨の変数名の付け方もあります。これを知っておくとヘルプを見なくてもどのようなものか何となくわかるようになります。興味のある人は公式マニュアルを見てみましょう。

```
In [16]: x = 10
```

```
Out[16]: 10
```

```
In [17]: x # x の中身を確認して実際に 10 が入っていることを確認してみましょう。
```

```
Out[17]: 10
```

変数を利用すると下記のようにコードの再利用性が上がります。

```
In [18]: a = 10 # a, b, c の値を変えて実行してみましょう。
         b = 3
         c = 90
         (a * b * c) / (a * b + b * c + c * a)
```

```
Out[18]: 2.25
```

```
In [19]: π # π には始めから円周率の値が入っている
```

```
Out[19]: π = 3.1415926535897...
```

```
In [20]: 田中 = 10
         太郎 = 3
         田中 * 太郎
```

```
Out[20]: 30
```

```
In [ ]:
```

目次に戻る

4.2.1 変数の更新

先程は変数 x に 10 という値を代入しましたが、x = 100 とすると変数の値を 100 に更新することが出来ます。

```
In [21]: x = 10
```

```
Out[21]: 10
```

```
In [22]: x = 100
```

```
Out[22]: 100

In [23]: x # 値が 100 に更新された

Out[23]: 100
```

現在の変数 x の値を、2 倍したものに更新するにはどのようにしたらよいでしょうか？わざわざ変数の値を確認して

```
x = 2 * 100
```

としていては面倒この上ありません。
このような場合には

```
x = 2 * x
```

とすることによって変数の値を更新することが出来ます。

```
In [24]: x

Out[24]: 100

In [25]: x = 2 * x

Out[25]: 200

In [26]: x

Out[26]: 200
```

掛け算だけでなく、その他の演算に対しても同様の文法で値を更新することが出来ます。

```
In [27]: x = x + 2

Out[27]: 202

In [28]: x = x / 2

Out[28]: 101.0
```

上の計算例では

```
x = 2 * x
```

と書きましたが、全く同じ意味で

```
x *= 2
```

と書くことも出来ます。このような書き方は慣れるまでに時間がかかるかもしれませんが、プログラミングではこのように書くほうがより一般的なので早く慣れましょう。

x = x + a	x += a
x = x - a	x -= a
x = x * a	x *=a | | x = x/a | x /= a |

```
In [29]: x = 10
         a = 2
         x += a

Out[29]: 12

In [ ]:
```

目次に戻る

5　比較演算子

比較演算子は 2 つの値を比較するときに使う演算子です。名前のまんまですね。比較演算子には以下のようなものが有ります。

Operator	Name
==	equality
!=, ≠	inequality
<	less than
<=, ☒	less than or equal to
>	greater than
>=, ☒	greater than or equal to

実際に使ってみてどのような結果が返ってくるのか試してみましょう。

```
In [30]: 1 < 3

Out[30]: true

In [31]: 10 < 5

Out[31]: false

In [32]: 2.5 <= 2.5

Out[32]: true

In [33]: x, y = 10, 10.0
         x == y

Out[33]: true

In [ ]:
```

目次に戻る

6　論理演算

P	Q	P ∧ Q	P ∨ Q	¬ P ∧ Q	P ∨¬ Q	¬ (P ∧ Q)	¬ P ∨¬ Q	¬ (P ∨ Q)	¬ P ∧¬ Q	P ∧¬ Q
T	T	T	T	F	T	F	F	F	F	F
T	F	F	T	F	T	T	T	F	F	T
F	T	F	T	T	F	T	T	F	F	F
F	F	F	F	F	T	T	T	T	T	F

Julia において「かつ (and)」は " && “, 「または (or)」は” | | “, 否定は真偽の前に” ! " をつけます。実際にいろいろ試してみて上の真偽表と同じになるか確認してみましょう。

```
In [34]: true
```

```
Out[34]: true
```

```
In [35]: !true
```

```
Out[35]: false
```

```
In [36]: true && true
```

```
Out[36]: true
```

```
In [37]: true && false
```

```
Out[37]: false
```

```
In [38]: true || false
```

```
Out[38]: true
```

```
In [39]: 1 < 3 && 3 < 5
```

```
Out[39]: true
```

Julia 以外のプログラミング言語では $a < x < b$ を表現する場合、

```
a < x && x < b
```

のように 2 つの条件式を && でつなぐ必要が有りますが、Julia では $a < x < b$ のまま使えるため、より数学的に表現することが出来ます。

```
In [40]: x = 3
         1 < x < 5
```

```
Out[40]: true
```

```
In [ ]:
```

目次に戻る

6.1 文字列

今までは変数に数字を代入してきましたが、変数に代入できるものは数字に限りません。変数には後述する配列や関数などあらゆる物を代入することが出来ます。

数字以外の例としてここでは変数に文字列を代入してみましょう。Julia では " " で囲った部分が文字列になります。

```
In [41]: str = "Hello!"
```

```
Out[41]: "Hello!"
```

```
In [42]: str
```

```
Out[42]: "Hello!"
```

```
In [ ]:
```

文字列 1 * 文字列 2

とすると文字列 1, 2 が連結されます。

```
In [43]: str1 = "Hello "
         str2 = "World!"
         str1 * str2
```

```
Out[43]: "Hello World!"
```

```
In [ ]:
```

べき乗にすると繰り返しになります

```
In [44]: "トゥー"^8 * " どこまで行くの 僕達今夜"
```

```
Out[44]: "トゥートゥートゥートゥートゥートゥートゥートゥー どこまで行くの 僕達今夜"
```

```
In [ ]:
```

文字列の中に変数の値を入れたい場合は、$変数名 とします。

```
In [45]: today = Dates.today()
         "今日は $today です"
```

```
Out[45]: "今日は 2017-03-15 です"
```

```
In [ ]:
```

目次に戻る

6.2 練習問題

6.2.1 1.1

下記のコードの出力結果はどれか?

```
x = 5
print("x")
```

- x
- 5

- “x”
- “5”

In [ ]:

6.2.2 1.2

下記のコードの出力結果はどれか?

```
x = 5
print("$x")
```

- x
- 5
- “$x”
- “$5”

In [ ]:

6.2.3 2.

a, b の値はいくつか?

```
a = 3
b = 7
b = a
println(a, ", ", b)
```

- (a, b) = (3, 7)
- (a, b) = (7, 3)
- (a, b) = (3, 3)
- (a, b) = (7, 7)

In [ ]:

6.2.4 3.

c の値はいくつになるか?

```
a, b, c = 18, 9, 2
c += a
c *= b
```

In [ ]:

6.2.5 4.

rem, mod の答えが一致する組み合わせはどれか? - rem(201, 5), mod(-201, 5) - rem(201, 5), mod(201, 5) - rem(-19, 10), mod(-19, 10) - rem(19, -10), mod(19, -10)

In [ ]:

6.2.6 5.

```
a = -1
b = 10
```

の時、rem を使って mod(a,b) と同じ結果を得るためにはどうしたら良いか？

```
In [ ]:
```

目次に戻る