Переменные и типы данных

Переменные

Переменная в **Julia** – это имя, связанное со значением. Ее целесообразно применять, когда вы хотите сохранить значение (полученное, например, после математических вычислений) для последующего использования.

Например, присвоим переменной x значение 10. Это делается с помощью **оператора присваивания** =.

[ ]:
```
x = 10
```

Для добавления текстового комментария служит символ # (решетка). Весь текст после # и до конца строки будет рассматриваться не как часть кода, а как комментарий. Например:

[ ]:
```
x = 10   # Присвоим переменной x значение 10
```

Если в строке кода набрать просто имя переменной, то после выполнения строки на экран будет выведено значение этой переменной (конечно, если переменной ранее было присвоено какое-нибудь значение).

[ ]:
```
x
```

Переменную x можно использовать в дальнейших вычислениях:

[ ]:
```
z = x + 5
```

Переменной x можно присваивать новые значения. Например, $\sqrt{2}$ или “Hello”:

[ ]:
```
x = sqrt(2)
```

[ ]:
```
x = "Hello"
```

Язык **Julia** чувствителен к регистру, т.е. переменные с именами a и A являются разными, функции с названиями Func и func тоже различны.

Текст в **Julia** вводится с использованием кодировки UTF-8. В кодировке UTF-8 символы могут быть представлены разным количеством байтов.

✏Задание 1

Присвойте переменной y значение 12.75. Напомним, что десятичная запятая в **Julia** обозначается точкой.

Затем измените значение переменной y, присвоив ей значение -23.842.

[ ]:
```

```

Решение

[ ]:
```
y = 12.75
```

[ ]:
```
y = -23.842
```

Разрешенные имена переменных

Имена переменных должны начинаться с буквы (A – Z или a – z), цифры, знака подчеркивания (_) или подмножества кодов символов Unicode больше 00A0. Последующие символы могут также включать !, цифры и другие символы. Таким образом, в именах переменных могут присутствовать буквы кириллицы, греческие буквы, различные математические символы и т.д.

Примеры:

[ ]:
```
Радиус = 100
```

[ ]:
```
φ = 1.75
```

В именах переменных нельзя использовать пробелы.

Запрещенными именами для переменных являются имена встроенных **ключевых слов**.

Ниже приведен список зарезервированных ключевых слов в **Julia**: `baremodule`, `begin`, `break`, `catch`, `const`, `continue`, `do`, `else`, `elseif`, `end`, `export`, `false`, `finally`, `for`, `function`, `global`, `if`, `import`, `let`, `local`, `macro`, `module`, `quote`, `return`, `struct`, `true`, `try`, `using`, `while`. Их запрещено использовать в качестве имен переменных.

Также зарезервированы следующие последовательности из двух слов: `abstract type`, `mutable struct`, `primitive type`. Однако можно создавать переменные с именами `abstract`, `mutable`, `primitive` и `type`.

Больше об именах переменных вы можете прочитать в документации.

Специальные символы Unicode можно вводить **в командной строке Engee** следующим образом: сначала ввести обратный слэш \, затем имя символа LaTeX и затем нажать клавишу табуляции (TAB).

Например, имя переменной α можно ввести, набрав **в командной строке Engee** `\alpha` + TAB:

```
α = 0.001
```

А знак квадратного корня можно ввести, набрав `\sqrt` + TAB.

Если вы где-то нашли символ, например в чужом коде, и не знаете, как его ввести, обратитесь к справке: **в командной строке Engee** введите ?, а затем вставьте символ. Например:

```
help?> α
"α" can be typed by \alpha<tab>
```

✏Задание 2

Перейдите в **командную строку Engee** и задайте переменной β значение 0.1. После выполнения задания вернитесь в **редактор скриптов**.

Подсказка Введите **в командной строке** `\beta + TAB`. После этого введите `= 0.1` и нажмите клавишу `Enter`.

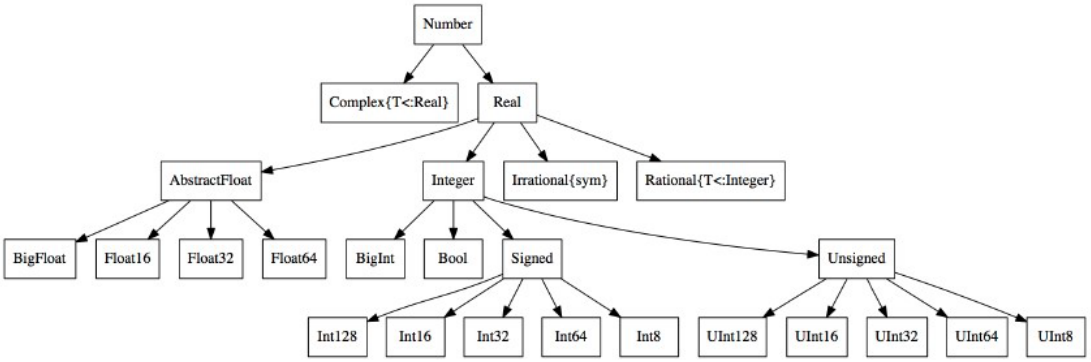
Решение

```
β = 0.1
```

Типы данных

Каждая величина обязательно имеет какой-то тип, но связан он не с именами переменных (как это характерно для предварительно компилируемых языков вроде **C/C++**), а с самими величинами, поэтому переменная не требует обязательного объявления типа: он станет известен после присваивания ей величины.

Типы данных в **Julia** образуют стройную древовидную иерархию. Самый верхний (общий) тип – `Any`. Все остальные типы являются его подтипами (субтипами). Тип `Number` (число) является подтипом `Any`. В свою очередь, `Number` является супертипом для `Real` и `Complex`. Иерархия подтипов `Number` в **Julia** приведена на рисунке.



Целые числа – Integer

Целый тип `Integer` – это подтип вещественного типа `Real`. В свою очередь, подтипами `Integer` являются `Int8`, `Int16`, `Int32`, `Int64`, `Int128` (со знаком) и `UInt8`, `UInt16`, `UInt32`, `UInt64`, `UInt128` (без знака). Здесь 8, 16, 32, 64, 128 – число бит, которое занимает переменная в памяти.

Наименьшее и наибольшее значения, которые может принять переменная, можно получить с помощью функций `typemin()` и `typemax()` соответственно.

Предельные параметры целочисленных типов **Julia** демонстрирует следующий пример:

[ ]:
```
for T in␣
  ↪[Int8,Int16,Int32,Int64,Int128,UInt8,UInt16,UInt32,UInt64,UInt128]
     println("$(lpad(T,7)): [$(typemin(T)),$(typemax(T))]")
end
```

В результате выполнения этого кода будут выведены следующие данные:

```
  Int8: [-128,127]
  Int16: [-32768,32767]
  Int32: [-2147483648,2147483647]
  Int64: [-9223372036854775808,9223372036854775807]
 Int128: [-170141183460469231731687303715884105728,170141183460469231731687303715884
  UInt8: [0,255]
 UInt16: [0,65535]
 UInt32: [0,4294967295]
 UInt64: [0,18446744073709551615]
UInt128: [0,340282366920938463463374607431768211455]
```

Примеры задания переменных целого типа:

[ ]:
```
a = 100
```

[ ]:
```
b = -232143284
```

Логические переменные – Bool

Логический тип является подтипом `Integer`. Логические переменные типа `Bool` также являются 8-битными целыми числами. Значение `true` (истина) соответствует 1, значение `false` (ложь) соответствует 0.

Например:

[ ]:
```
a = true
```

[ ]:
```
b = false
```

[ ]:
```
c = 1 > 2
```

[ ]:
```
d = 1 < 2
```

Переменная c принимает значение `false`, т.к. логическое выражение `1 > 2` ложное. А переменная d принимает значение `true`, т.к логическое выражение `1 < 2` истинное. Подробнее о логических выражениях мы поговорим в Главе 03 **Выражения**.

Вещественные числа

Вещественнный тип `AbstractFloat` содержит следующие типы: `Float16`, `Float32` и `Float64`. Число в названии типа показывает количество байтов, занимаемых в памяти на хранение переменной. Например:

[ ]:
```
a = 1.0
```

[ ]:
```
b = -12.75
```

Можно использовать экспоненциальную нотацию. Число $1,6 \cdot 10^{-19}$ вводится так:

[ ]:
```
x = 1.6e-19
```

По умолчанию приведенные выше вещественные числа имеют тип `Float64`. Если вместо `e` ввести `f`, то получится число в формате `Float32`:

[ ]:
```
x = 1.6f-19
```

Число π задается с помощью зарезервированной константы `pi`:

[ ]:
```
x = pi
```

✏Задание 3

Задайте следующие вещественные переменные: $s = -324,83424$, $t = 6,022 \cdot 10^{23}$ и $h = \pi/4$.

[ ]:
```

```

Решение

[ ]:
```
s = -324.83424
```

[ ]:
```
t = 6.022e23
```

[ ]:
```
h = pi/4
```

Комплексные числа – Complex

Глобальная константа `im` обозначает мнимую единицу i (главное значение квадратного корня из -1). **Комплексные числа** записываются в виде `a+b im`, где `a` и `b` – вещественные числа. Например: `3+2im`, `-1-5im` и т.д. Обратите внимание, что знак умножения * между мнимой частью и константой `im` можно не писать.

Примеры выполнения арифметических операций с комплексными числами:

[ ]:
```
(5 + 3im)*(4 - 2im)
```

[ ]:
```
(8 + 7im)/(3 - 4im)
```

[ ]:
```
(-9 + 1im) - (2 - 5im)
```

[ ]:
```
(3+7im)^3
```

[ ]:
```
(1+1im)^(1+1im)
```

Для работы с комплексными числами существуют следуюшие функции: * `real(z)` – вычисление вещественной части; * `imag(z)` – вычисление мнимой части числа; * `reim(z)` – вычисление вещественной и мнимой части в виде кортежа; * `conj(z)` – комплексное сопряжение числа; * `abs(z)` – вычисление модуля; * `abs2(z)` – вычисление квадрата модуля; * `angle(z)` – вычисление аргумента.

Примеры:

[ ]:
```
z = 2+3im
real(z)
```

[ ]:
```
imag(z)
```

[ ]:
```
reim(z)
```

[ ]:
```
conj(z)
```

[ ]:
```
abs(z)
```

[ ]:
```
abs2(z)
```

[ ]:
```
angle(z)
```

Для комплексных чисел также определны все элементарные функции. Например, мы можем вычислить e^z и $\sin z$:

[ ]:
```
exp(z)
```

[ ]:
```
sin(z)
```

✏Задание 4

Задайте комплексные числа $u = 6-8i$ и $v = 4+7i$. Вычислите сумму и произведение этих чисел. Для числа u найдите модуль и аргумент.

[ ]:
```

```

Решение

[ ]:
```
u = 6-8im
```

[ ]:
```
v = 4+7im
```

[ ]:
```
u + v
```

[ ]:
```
u * v
```

[ ]:
```
abs(u)
```

[ ]:
```
angle(v)
```

Если результат выполнения действия над вещественным числом принимает комплексное значение, то это вещественное число нужно предварительно преобразовать в комплексный тип с помощью функции `complex`.

Например, извлечение квадратного корня из отрицательного числа выполнятеся так:

[ ]:
```
sqrt(complex(-2))
```

А вот пример вычисления натурального логарифма от отрицательного числа:

[ ]:
```
log(complex(-1))
```

Если аргументы этих функций не преобразовать в комплексный вид, то возникнет ошибка:

```
sqrt(-2)

DomainError with -2.0:
sqrt will only return a complex result if called with a complex argument. Try sqrt

log(-1)

DomainError with -1.0:
log will only return a complex result if called with a complex argument. Try log(C
```

Функцию `complex` также можно использовать для формирования комплексного числа напрямую из его вещественной и мнимой частей:

[ ]:
```
complex(4,5)
```

Рациональные числа – Rational

В **Julia** реализована интересная возможность – вычисления с **рациональными числами**. Рациональные числа записываются в виде дроби вида `a//b`, где `a` и `b` – целые числа.

Например, рациональная дробь $4/9$ записывается следующим образом:

[ ]:
```
4//9
```

При сложении и вычитании рациональных чисел они приводятся к общему знаменателю, их числители складываются (вычитаются) и затем, если это возможно, выполняется сокращение дроби. Например:

[ ]:
```
1//3 + 1//7
```

Преобразовать рациональное число в вещественный тип можно с помощью функции `float`:

[ ]:
```
a = 3//8
float(a)
```

✏Задание 5

Найдите точное значение суммы рациональных дробей $5/17 + 8/19$.

[ ]:
```

```

Подсказка При записи рациональной дроби числитель и знаменатель разделяются знаком `//`.

Решение

[ ]:
```
5//17 + 8//19
```

Символы – Char

Переменные **символьного типа** `Char` задаются следующим образом:

[ ]:
```
c = 'A'
```

Строки – String

Строка в **Julia** – это упорядоченный набор символов. Строки заключаются в двойные кавычки " " или в тройные двойные кавычки """ """.

Пример:

[ ]:
```
s = "Hello, world!"
```

Подробнее о символьном и строковом типах данных мы поговорим в Главе 06 **Символы и строки**.

✏Задание 6

Задайте переменную строкового типа `s` и присвойте ей значение **Язык Julia**.

[ ]:
```

```

Решение

[ ]:
```
s = "Язык Julia"
```

Векторы – Array

Массивы (**упорядоченные** множества объектов одного типа) в **Julia** могут быть заданы как в виде **векторов** (`Array`), так и в виде **матриц** (`Matrix`).

Вектор (**одномерный массив**) задается перечислением его элементов через запятую или точку с запятой в квадратных скобках. Например:

[ ]:
```
a = [1, 2, 3, 4, 5]
```

[ ]:
```
a = [1; 2; 3; 4; 5]
```

Матрицы – Matrix

Матрица-строка (**одномерный массив**) создается с помощью перечисления ее элементов через пробел в квадратных скобках. Например:

[ ]:
```
a = [1 2 3 4 5]
```

Если матрица содержит несколько строк (**двумерный массив**), то она задается аналогично. Ввод элементов осуществляется по строкам, строки отделяются друг от друга точкой с запятой. Например:

[ ]:
```
a = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15]
```

Разумеется, элементами векторов и матриц могут служить не только целые числа, как в приведенных примерах, но и элементы любого другого типа (вещественные, комплексные, рациональные числа, символы, строки и даже другие векторы и матрицы).

Подробнее о векторах и матрицах мы поговорим в Главе 05 **Массивы**.

✏Задание 7

Задайте матрицу `A`, состоящую из следующих элементов:

$$\begin{pmatrix} -1.3 & 4.0 & 8.1 \\ 4.2 & -9.1 & 10.4 \\ 7.3 & 3.8 & -0.6 \end{pmatrix}.$$

[ ]:
```

```

Подсказка Вводите элементы матрицы построчно: элементы одной строки должны отделяться пробелами, а строки должны отделяться точкой с запятой.

Решение

[ ]:
```
A = [-1.3 4.0 8.1; 4.2 -9.1 10.4; 7.3 3.8 -0.6]
```

Кортежи – Tuple

Кортеж (Tuple) – группа значений некоторых величин фиксированной длины, разделенных запятыми, иногда эту группу окружают круглыми скобками. Кортеж может содержать данные **разных типов**, в отличие от массива, в котором содержатся данные одного типа.

Пример:

[ ]:
```
c = (1, 1.1, pi, 'c', "Julia", 1//3)
```

Кортеж можно превратить в вектор типа Any: v=[c…].

Вектор можно превратить в кортеж: c=(v…,).

Элементы кортежа изменять нельзя, в отличие от элементов массива. Выражение c[1] = 2 вызовет сообщение об ошибке.

Можно использовать именованный кортеж вида c = (a=1, b=2.2). Доступ к элементам кортежа возможен либо через индекс, например, c[1], либо по имени c.a, c.b.

Если список возвращаемых величин какой-либо функции содержит несколько значений, то возвращаются они в форме кортежа или массива (когда возвращается массив данных).

Множества – Set

Множества (в отличие от массивов) используются для хранения коллекций **неупорядоченных** уникальных значений. Множество в **Julia** создается с помощью команды Set. В качестве аргумента в квадратных скобках записывается массив (вектор или матрица) элементов, из которых должно состоять множество.

Пример. Создадим множество, состоящее из элементов 1, 3, 5 и 7:

[ ]:
```
s = Set([1, 3, 5, 7])
```

Добавление элемента к множеству производится с использованием оператора push!.

Пример. Добавим в множество s элемент 2:

[ ]:
```
push!(s, 2)
```

Другие функции для работы с множествами: * intersect(set1,set2) – пересечение; * union(set1,set2) – объединение; * setdiff(set1,set2) – разность.

Превратить строку st в множество содержащихся в ней символов можно так: x = Set(st).

Аналогичным образом в множество можно превратить массив z (одномерный или многомерный): `x = Set(z)`.

✏Задание 8

Задайте множество s, состоящее из элементов символьного типа: ‘A’, ‘B’, ‘C’ и ‘D’. Затем добавьте в множество s новый элемент ‘E’.

[ ]:

Подсказка Чтобы задать множество, используйте команду Set. Затем в круглых скобках запишите вектор, состоящий из элементов этого множества. Напомним, что элементы вектора заключаются в квадратные скобки. Для добавления нового элемента в множество используйте функцию push!.

Решение

[ ]:
```
s = Set(['A', 'B', 'C', 'D'])
```

[ ]:
```
push!(s, 'E')
```

Словари – Dict

Словарь (Dict) – это ассоциативный массив, состоящий из пар **ключ – значение**. Словарь создается с помощью оператора Dict.

Примеры задания словаря:

[ ]:
```
d = Dict('a'=>1, 'b'=>2, 'c'=>3)
```

[ ]:
```
d = Dict([('a', 1), ('b', 2), ('c', 3)])
```

Здесь 'a', 'b' и 'c' – имена ключей. В данном примере они заданы переменными символьного типа. А 1, 2 и 3 – значения этих ключей. Имена ключей могут быть символами или строками.

Константы

Поскольку тип переменных в **Julia** легко изменяется, это приводит к дополнительной нагрузке на ресурсы компьютера. Чтобы уменьшить эту нагрузку, можно использовать определение переменной с использованием ключевого слова const:

[ ]:
```
const M = 10.0
```

[ ]:
```
const N = [1 2 3]
```

Значения переменных, объявленных таким образом, можно изменять, однако их тип менять нельзя, т.е. `M = 5.0` допустимо, а `M = [1 2 3]` – нет.

Определение типа величины

Определить тип какой-либо величины можно, вызывая для нее функцию `typeof()`.

Примеры:

```
[ ]: typeof(10)
```

```
[ ]: typeof(3.14)
```

```
[ ]: typeof("name")
```

```
[ ]: typeof([1; 2; 3])
```

```
[ ]: typeof(true)
```

✏Задание 9

С помощью функции `typeof` определите тип следующих величин: * `3+5im`; * `'Q'`; * `123.0` (без кавычек); * `"123.0"` (в двойных кавычках).

```
[ ]:
```

Решение

```
[ ]: typeof(3+5im)
```

```
[ ]: typeof('Q')
```

```
[ ]: typeof(123.0)
```

```
[ ]: typeof("123.0")
```

Преобразование типов

Запись `T(x)` или `convert(T,x)` преобразует величину `x` в значение типа `T`.

Примеры преобразования вещественного типа в целый (команда `Int` равносильна преобразованию значения в тип `Int64`):

```
[ ]: Int(127.0)
```

```
[ ]: convert(Int, 127.0)
```

Если дробная часть вещественнного числа отлична от нуля, то при попытке его преобразования в целый тип возникнет ошибка. Например:

`Int(3.14)`

```
ERROR: InexactError: Int64(3.14)
Stacktrace:
 [1] Int64(x::Float64)
   @ Base ./float.jl:900
 [2] top-level scope
   @ In[29]:1
```

Рассмотрим еще несколько примеров.

Преобразование логического типа в целый:

[ ]:
```
Int(true)
```

Преобразование логического типа в вещественный (команда `float` равносильна преобразованию значения в тип `Float64`):

[ ]:
```
float(false)
```

Преобразование целого типа в вещественный:

[ ]:
```
float(100)
```

Преобразование рационального типа в вещественный:

[ ]:
```
float(137//693)
```

Преобразование целого типа в комплексный:

[ ]:
```
Complex(2)
```

Преобразование вещественного типа в комплексный:

[ ]:
```
Complex(25.1)
```

Преобразование целого типа в строковый:

[ ]:
```
string(100)
```

Преобразование вещественного типа в строковый:

[ ]:
```
string(123.45)
```

Преобразование символьного типа в строковый:

[ ]:
```
string('A')
```

✏Задание 10

Преобразуйте: * вещественное число `-327.0` в целый тип; * рациональную дробь `1//9` в вещественный тип; * целое число `-17` в комплексный тип; * вещественное число `2.718` в строковый тип.

[ ]:

Решение

[ ]:
```
Int(-327.0)
```

[ ]:
```
float(1//9)
```

[ ]:
```
Complex(-17)
```

[ ]:
```
string(2.718)
```

Чтобы преобразовать переменную строкового типа x в числовой тип T (целый, вещественный), используется функция parse:

```
parse(T, x)
```

Пример. Преобразуем строку в целый тип:

[ ]:
```
parse(Int, "123")
```

Преобразуем строку в вещественный тип (здесь нужно указывать конкретный тип вещественных чисел: Float16, Float32 или Float64):

[ ]:
```
parse(Float64, "123.45")
```

Особые значения с плавающей запятой

Существует три определенных стандартных значения с плавающей запятой, которые не соответствуют ни одной точке на прямой вещественных чисел: * Inf – положительная бесконечность. Значение, большее чем все конечные значения с плавающей запятой. * -Inf – отрицательная бесконечность. Значение, меньшее чем все конечные значения с плавающей запятой. * NaN (Not a Number) – не число. Значение, не равное никакому значению с плавающей запятой (включая его само).

Эти значения являются результатами определенных арифметических операций:

[ ]:
```
1/0
```

[ ]:
```
-5/0
```

[ ]:
```
1/Inf
```

[ ]:
```
1/-Inf
```

[ ]:
```
0/0
```

[ ]:
```
Inf - Inf
```

[ ]:
```
0 * Inf
```

Тест для получения сертификата

Пройти тест по теме **Переменные и типы данных**.