Символы и строки

Символы

Встроенным типом, используемым для **символов** в **Julia**, является тип `Char`. Значение типа `Char` представляет собой один символ. Это 32-разрядный примитивный тип, который может быть преобразован в числовое значение, представляющее код символа Unicode. Значения типа `Char` вводятся и отображаются в **одинарных кавычках**:

[ ]:
```
c = 'x'
```

Переменные и константы типа `Char` с помощью функции `Int` можно преобразовать в целочисленное значение, т.е. в код символа:

[ ]:
```
c = Int('x')
```

[ ]:
```
typeof(c)
```

✏Задание 1

Задайте переменные символьного типа `x`, `y` и `z`, присвоив им значения букв русского алфавита А, Б и В. Преобразуйте их в целочисленные значения.

[ ]:
```

```

Решение

[ ]:
```
x = 'А'
```

[ ]:
```
y = 'Б'
```

[ ]:
```
z = 'В'
```

[ ]:
```
Int(x)
```

[ ]:
```
Int(y)
```

[ ]:
```
Int(z)
```

Целочисленное значение с помощью функции `Char` можно преобразовать обратно в символьный тип:

[ ]:
```
Char(120)
```

Вы можете выполнять сравнения и арифметические действия со значениями типа `Char` (на самом деле все действия выполняются с кодами этих символов):

```
[ ]: 'A' < 'a'
```

```
[ ]: 'A' <= 'a' <= 'Z'
```

```
[ ]: 'x' - 'y'
```

```
[ ]: 'A' + 1
```

✏Задание 2

Выполните сравнение символьных значений `'x'` и `'y'` с помощью операторов >, <, == и !=.

```
[ ]:
```

Решение

```
[ ]: 'x' > 'y'
```

```
[ ]: 'x' < 'y'
```

```
[ ]: 'x' == 'y'
```

```
[ ]: 'x' != 'y'
```

Не все целочисленные значения являются допустимыми кодами символов Unicode, но для повышения производительности преобразование типа `Char` не проверяет допустимость каждого символьного значения. Чтобы проверить тот факт, что каждое преобразованное значение является допустимым кодом символа, используйте функцию `isvalid`. Например:

```
[ ]: Char(0x110000)
```

```
[ ]: isvalid(Char, 0x110000)
```

Допустимыми кодами символов Unicode являются U+0000–U+D7FF и U+E000–U+10FFFF. Еще не всем им присвоены понятные значения, и они не обязательно интерпретируются приложениями, но все эти значения считаются допустимыми символами Unicode.

Вы можете ввести любой символ Unicode в одинарных кавычках, используя символ `\u`, за которым следует до четырех шестнадцатеричных цифр, или символ `\U`, за которым следует до восьми шестнадцатеричных цифр (для самого длинного допустимого значения требуется только шесть цифр).

```
[ ]: '\u78'
```

[ ]:
```
'\u2200'
```

[ ]:
```
'\U10ffff'
```

Строки

Строки – это конечные последовательности символов.

Есть несколько примечательных особенностей, касающихся строк **Julia**.

- Встроенным типом, используемым для строк в **Julia**, является тип `String`. Он поддерживает полный набор символов Unicode через кодировку UTF-8. Для преобразования в другие кодировки Unicode и из них существует функция `transcode`.
- Строки являются неизменяемыми: значение объекта `String` не может быть изменено. Чтобы создать другое строковое значение, нужно построить новую строку из частей других строк.

Строковые значения вводятся в **двойных кавычках** или в **тройных двойных кавычках**. Отображаются они всегда в двойных кавычках.

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

[ ]:
```
s = """Выражение "в кавычках" внутри строки"""
```

Длинные строки можно разбить на несколько строк, предваряя новую строку обратным слешем (\).

[ ]:
```
s = "This is a long \
     line"
```

Чтобы извлечь символ из строки, к нему нужно обратиться по индексу, как к элементу одномерного массива.

Например, извлечем первый элемент из заданной ранее строки `s`:

[ ]:
```
s[begin]
```

Шестой элемент:

[ ]:
```
s[6]
```

Последний элемент:

[ ]:
```
s[end]
```

Индекс первого элемента (первый символ строки) возвращается функцией `firstindex(s)`, а индекс последнего элемента (символа) – функцией

`lastindex(s)`. Ключевые слова `begin` и `end` могут использоваться внутри операции индексирования как сокращение для первого и последнего индексов соответственно в заданном измерении. Индексирование строк, как и большинство индексирований в **Julia**, начинается с 1. Функция `firstindex` всегда возвращает 1 для любого объекта типа `String`. Однако функция `lastindex(s)` обычно не то же самое, что функция `length(s)` для строки, поскольку некоторые символы Unicode могут занимать несколько единиц кода.

✏Задание 3

Задайте строковую переменную со значением **Programming on Julia**. Затем извлеките из нее второй, шестнадцатый и предпоследний символы.

[ ]:

Решение

[ ]:
```
s = "Programming on Julia"
```

[ ]:
```
s[2]
```

[ ]:
```
s[16]
```

[ ]:
```
s[end-1]
```

Чтобы извлечь подстроку, используется индексирование диапазона (оператор `:`):

[ ]:
```
s = "This is a long \
     line"
s[11:14]
```

Обратите внимание, что выражения `s[k]` и `s[k:k]` не дают одинакового результата.

[ ]:
```
s[4]
```

[ ]:
```
s[4:4]
```

В первом случае получается символьное значение типа `Char`, а во втором случае – строковое значение типа `String`, которое содержит только один символ. В **Julia** это совершенно разные вещи.

Для создания копии выбранной части исходной строки можно воспользоваться функцией `SubString`. Например:

[ ]:
```
str = "long string"
```

[ ]:
```
substr1 = SubString(str, 1, 4)
```

[ ]:
```
substr2 = SubString(str, 6, 11)
```

✏Задание 4

Задайте строковую переменную со значением `"Programming on Julia"`. Создайте из нее две подстроки так, чтобы первая содержала слово `"Programming"`, а вторая – слово `"Julia"`. Выполните задание двумя способами: с помощью оператора `:` и с помощью функции `SubString`.

[ ]:
```

```

Решение

[ ]:
```
s = "Programming on Julia"
```

[ ]:
```
s[1:11]
```

[ ]:
```
SubString(s, 1, 11)
```

[ ]:
```
s[16:20]
```

[ ]:
```
SubString(s, 16, 20)
```

Операции со строками

Конкатенация

Одной из самых распространенных и полезных операций со строками является **конкатенация**.

Пример. Зададим две строковые переменные и объединим их с помощью функции `string`:

[ ]:
```
h = "Hello"
w = "world"
str = string(h, ", ", w, ".")
```

Другой способ конкатенации – использование оператора `*` вместо функции `string`.

[ ]:
```
str = h * ", " * w * "."
```

✏Задание 5

Задайте две строковые переменные со значениями `"Альберт"` и `"Эйнштейн"`. Выполните конкатенацию этих строк двумя укказанными способами так, чтобы в результате получилась строка со значением `"Альберт Эйнштейн"`.

[ ]:

Подсказка Между двумя словами нужно поставить пробел. Поэтому вам нужно включить в конкатенацию третью строку, содержащую символ пробела (`" "`).

Решение

[ ]:
```
a = "Альберт"
b = "Эйнштейн"
s = string(a, " ", b)
```

[ ]:
```
s = a * " " * b
```

Еще один способ объедиения строк – **интерполяция**. Для этого объединяемые строки записываются подряд в общих двойных кавычках, а перед именами строковых перменных ставится символ $:

[ ]:
```
str = "$h, $w."
```

С помощью символа $ можно преобразовать в строку результаты математических выражений. Например, результат вычисления выражения $4\pi^2$ преобразуется в строковый тип следующим образом:

[ ]:
```
s = "$(4pi^2)"
```

Это более удобочитаемая и целесообразная возможность, эквивалентная приведенной выше конкатенации строк. Система переписывает это выражение в вызов `string(h,  w, ".")`.

Чтобы включить символ $ в строку, экранируйте его обратным слешем:

[ ]:
```
print("I have \$100 in my account.")
```

Сравнение строк

Вы можете сравнивать строки, используя стандартные операторы сравнения.

[ ]:
```
"one" < "two"
```

[ ]:
```
"one" == "two"
```

[ ]:
```
"Hello" != "Goodbye"
```

✏Задание 6

Задайте две строковые переменные со значениями `"Microsoft"` и `"Apple"`. Выполните их сравнение с помощью операторов >, <, == и !=.

[ ]:

Решение

[ ]:
```
a = "Microsoft"
b = "Apple"
a > b
```

[ ]:
```
a < b
```

[ ]:
```
a == b
```

[ ]:
```
a != b
```

Поиск символа в строке

Вы можете искать индекс первого символа в строке с помощью функции `findfirst` и индекс последнего символа с помощью функции `findlast`:

[ ]:
```
findfirst('g', "programming")
```

[ ]:
```
findlast('g', "programming")
```

[ ]:
```
findfirst('b', "programming")
```

В третьем примере, при отсутствии в строке заданного символа, функция `findfirst` не выдала никакого результата.

✏Задание 7

С помощью функций `findfirst` и `findlast` найдите индексы первого и последнего символа `'i'` в строке `"Mississippi"`.

[ ]:

Решение

[ ]:
```
s = "Mississippi"
findfirst('i', s)
```

[ ]:
```
findlast('i', s)
```

Вы можете начать поиск символа с заданного смещения с помощью функций `findnext` и `findprev`.

```
[ ]: findnext('o', "xylophone", 1)
```

```
[ ]: findnext('o', "xylophone", 5)
```

```
[ ]: findprev('o', "xylophone", 5)
```

```
[ ]: findnext('o', "xylophone", 8)
```

Поиск подстроки в строке

Вы можете использовать функцию `occursin(substr, str)`, чтобы проверить, содержится ли подстрока `substr` в строке `str`.

```
[ ]: occursin("world", "Hello, world.")
```

```
[ ]: occursin("o", "Julia")
```

✏Задание 8

С помощью функции `occursin` проверьте, содержатся ли подстроки `"привет"` и `"тебя"` в строке `"Я пришел к тебе с приветом"`.

```
[ ]:
```

Решение

```
[ ]: s = "Я пришел к тебе с приветом"
occursin("привет", s)
```

```
[ ]: occursin("тебя", s)
```

Другие функции

Далее перечислены некоторые другие полезные функции для работы со строками.

- `length(str)` возвращает количество символов в строке (`str`).
- `length(str, i, j)` возвращает количество допустимых индексов символов в строке (`str`) от `i` до `j`.
- `thisind(str, i)` при заданном произвольном индексе в строке находит первый индекс символа, на который указывает индекс.

- `nextind(str, i, n=1)` находит начало n-го символа, начинающегося после индекса (`i`).
- `prevind(str, i, n=1)` находит начало n-го символа, начинающегося перед индексом (`i`).

✏Задание 9

С помощью функции `length` найдите общее количество символов в строке `"abracadabra"` и количество символов с индексами от 3-го до 5-го.

[ ]:

Решение

[ ]:
```
length("abracadabra")
```

[ ]:
```
length("abracadabra", 3, 5)
```

Больше о строках вы можете прочитать в документации.

Тест для получения сертификата

Пройти тест по теме **Символы и строки**.