Tutorial II.I - Variables and Types

Applied Optimization with Julia

Introduction

Welcome to this interactive Julia tutorial which introduces the basics of variables and types. Understanding variables and their types is crucial as they are the building blocks of any program. They determine how data is stored, manipulated, and how efficiently your code runs.

This script is designed to be interactive. Follow the instructions, write your code in the designated code blocks, and then execute the corresponding code. Each exercise is followed by an `@assert` statement that checks your solution.

Julia in VS Code

1. First, install the Julia and the Jupyter Extension from the extensions menu on the left side.
2. Install the Jupytext for Notebooks Extension
3. Open VS Code and navigate to the folder where you want to save all the files for this course.
4. Open the terminal in VS Code (Terminal → New Terminal) and run:

   ```
   julia --project=applied-optimization
   ```

   This starts Julia with the course project environment.
5. In the Julia REPL that opens, run:

   ```
   using Pkg
   Pkg.instantiate()  # Initialize the project
   Pkg.add("IJulia")  # Add IJulia to the environment
   ```

6. Still in Julia, run:

   ```
   using IJulia
   IJulia.installkernel("Applied Optimization", "--project=applied-
   optimization")
   ```

   This creates a Jupyter kernel that uses the course environment.
7. Type `exit()` or press Ctrl+D to exit the Julia REPL.

8. Download the first notebook from the course website and save it in your course folder. I recommend to download the `.jl` Julia files by clicking on Julia. If these don't work, download the `.ipynb` files by clicking on Jupyter on the course website.
9. If you work with `.jl` files, right-click on any dowloaded `.jl` file and select "Open with Jupyter Notebook". If you work with `.ipynb` files, just open the file and you are good to go.
10. Click the kernel selector (top-right corner) and choose "Applied Optimization" from the list.

> ⚠ Warning
>
> Sorry, that this start is rather complicated. But in following this, we have a clean environment we can work in and you basically cannot break anything with the installation of packages. If you use `.jl` files your files are saved as `.jl` file while also being a notebook which is great for Git version control (Git)! If you use `.ipynb`, it's fine as well, although version control of notebooks does not work so good.

> ℹ Note
>
> These steps ensure you're working in the correct Julia environment with all course dependencies. The `--project=.` flag tells Julia to use the `Project.toml` file in the current directory, keeping all packages organized and avoiding conflicts.

Section 1 - Variables

Think of variables as labeled containers. Just like you might label a box "Books" to store books, in programming we label our data with variable names. For example:

```
age = 30        # A box labeled "age" containing the number 30
```

```
30
```

```
name = "Tobias"  # A box labeled "name" containing the text "Tobias"
```

```
"Tobias"
```

Exercise 1.1 - Declare a Variable

Declare a variable named `x` and assign it the value `1`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert x == 1 "Check again, the value of x should be 1. Remember to assign
the value directly to x."
println("Great, you have correctly assigned the value $x to the variable
'x'.")
```

> **i** Note
>
> Always replace 'YOUR CODE BELOW' with your actual code.

Exercise 1.2 - Declare a String Variable

Declare a variable named `hi` and assign it the string `"Hello, Optimization!"`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert hi == "Hello, Optimization!" "Make sure the variable 'hi' contains
the exact string \"Hello, Optimization\"!"
println("Good, the variable 'hi' now states \"$hi\".")
```

Section 2 - Basic Types

Just like real containers come in different types (like boxes for books, refrigerators for food, etc.), variables in Julia have different types depending on what they store:

- Integers (Int): Whole numbers like `1`, `42`, `-10`
- Floats (Float64): Numbers with decimal points like `3.14`, `-0.5`
- Booleans (Bool): True/false values like `true`, `false`
- Strings (String): Text in quotes like `"Hello"`

You can check what type of "container" a variable is using `typeof()`. Try this:

```
age = 25
typeof(age)     # Will show Int64 (integer type)
```

```
Int64
```

```
price = 19.99
typeof(price)   # Will show Float64 (decimal number type)
```

```
Float64
```

Exercise 2.1 - Create an Integer Variable

Create an Integer variable `answerUniverse` and set it to `42`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert answerUniverse == 42 "The variable 'answerUniverse' should hold
42."
println("Great, the answer to all questions on the universe is
$answerUniverse now.")
```

Exercise 2.2 - Create a Float Variable

Create a Float variable `money` and set it to `1.35`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert money == 1.35 "The variable 'money' should hold the Float64 1.35."
println("Perfect, the you have stored $money in the variable 'money'.")
```

Exercise 2.3 - Create a Boolean Variable

Create a Boolean variable `isStudent` and set it to `true`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert isStudent == true "The variable 'isStudent' should be set to true."
println("Correct, you are a student now.")
```

Section 3 - Type Annotations and Inference

Sometimes we want to specify exactly what kind of "container" we want to use. In Julia, we can do this using type annotations:

```
temperature::Float64 = 98.6    # Specifically saying we want a decimal
number
```

```
98.6
```

```
count::Int64 = 100            # Specifically saying we want a whole number
```

```
100
```

Exercise 3.1 - Type Annotation

Declare a variable `y` with an explicit type annotation of `Int64` and assign it the value `5`.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert y == 5 && typeof(y) == Int64 "Make sure 'y' is of type Int64 and
has the value 5."
println("Great! You've created an Int64 variable 'y' with the value $y.")
```

Section 4 - String Interpolation

String interpolation is like filling in blanks in a sentence. Instead of writing:

```
name = "Tobias"
age = 30
# The hard way:
message = "My name is " * name * " and I am " * string(age) * " years old"
```

```
"My name is Tobias and I am 30 years old"
```

We can use the `$` symbol to insert variables directly into our text:

```
message = "My name is $name and I am $age years old"
```

```
"My name is Tobias and I am 30 years old"
```

It's like having a template where Julia automatically fills in the values for you! If you have paid attention to the previous excercise, you have already seen this in action. The following example illustrates this again:

```
language = "Julia"
println("I'm learning $language")
```

```
I'm learning Julia
```

Exercise 4.1 - String Interpolation

Create a string `message` that says `"y is [value of y]"` using string interpolation.

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert message == "y is 5" "Make sure your string includes the correct
value of y."
println("Excellent! Your interpolated string is: $message")
```

Conclusion

Congratulations! You have completed the first tutorial on Variables and Types. You've learned about the basics of variables, integers, floats, booleans, and strings. Continue to the next file to learn more.

Solutions

You will likely find solutions to most exercises online. However, I strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Bibliography