

Introduction to programming for data science

STAT 201

Arvind Krishna

9/20/22

Table of contents

Preface	4
I Python	5
1 Introduction to Jupyter Notebooks and programming in python	6
1.1 Jupyter notebook	6
1.1.1 Introduction	6
1.1.2 Writing and executing code	7
1.1.3 Saving and loading notebooks	7
1.1.4 Rendering notebook as HTML	8
1.2 In-class exercise	8
1.3 Python libraries	8
1.4 Debugging and errors	9
1.5 Terms used in programming	9
2 Variables, expressions and statements	10
2.0.1 Practice exercise 1	10
2.1 Constants and Variables	11
2.1.1 Variable names	12
2.1.2 Practice exercise 2	13
2.2 Assignment statements	13
2.3 Expressions	14
2.3.1 Practice exercise 3	14
2.4 Converting datatypes	15
2.5 User input	16
2.5.1 Practice exercise 4	17
2.6 Commenting code	17
2.6.1 Practice exercise 5	17
2.7 Programming errors	17
2.7.1 Syntax errors	18
2.7.2 Run-time errors	18
2.7.3 Semantic errors	18
2.7.4 Practice exercise 6	19
2.8 Practice exercise 7	19

Appendices	19
A Datasets	20

Preface

This book is currently being written for the course STAT201.

Part I

Python

1 Introduction to Jupyter Notebooks and programming in python

This chapter is a very brief introduction to python and Jupyter notebooks. We only discuss the content relevant for applying python to analyze data.

Anaconda: If you are new to python, we recommend downloading the [Anaconda installer](#) and following the instructions for installation. Once installed, we'll use the Jupyter Notebook interface to write code.

Quarto: We'll use Quarto to publish the `**ipynb*` file containing text, python code, and the output. Download and install Quarto from [here](#).

1.1 Jupyter notebook

1.1.1 Introduction

Jupyter notebook is an interactive platform, where you can write code and text, and make visualizations. You can access Jupyter notebook from the Anaconda Navigator, or directly open the Jupyter Notebook application itself. It should automatically open up in your default browser. The figure below shows a Jupyter Notebook opened with Google Chrome. This page is called the *landing page* of the notebook.

<IPython.core.display.Image object>

To create a new notebook, click on the **New** button and select the **Python 3** option. You should see a blank notebook as in the figure below.

<IPython.core.display.Image object>

1.1.2 Writing and executing code

Code cell: By default, a cell is of type *Code*, i.e., for typing code, as seen as the default choice in the dropdown menu below the *Widgets* tab. Try typing a line of python code (say, `2+3`) in an empty code cell and execute it by pressing *Shift+Enter*. This should execute the code, and create an new code cell. Pressing *Ctrl+Enter* for *Windows* (or *Cmd+Enter* for *Mac*) will execute the code without creating a new cell.

Commenting code in a code cell: Comments should be made while writing the code to explain the purpose of the code or a brief explanation of the tasks being performed by the code. A comment can be added in a code cell by preceding it with a `#` sign. For example, see the comment in the code below.

Writing comments will help other users understand your code. It is also useful for the coder to keep track of the tasks being performed by their code.

```
#This code adds 3 and 5
3+5
```

8

Markdown cell: Although a comment can be written in a code cell, a code cell cannot be used for writing headings/sub-headings, and is not appropriate for writing lengthy chunks of text. In such cases, change the cell type to *Markdown* from the dropdown menu below the *Widgets* tab. Use any markdown cheat sheet found online, for example, [this one](#) to format text in the markdown cells.

Give a name to the notebook by clicking on the text, which says ‘Untitled’.

1.1.3 Saving and loading notebooks

Save the notebook by clicking on **File**, and selecting **Save as**, or clicking on the **Save and Checkpoint** icon (below the **File** tab). Your notebook will be saved as a file with an extension *ipynb*. This file will contain all the code as well as the outputs, and can be loaded and edited by a Jupyter user. To load an existing Jupyter notebook, navigate to the folder of the notebook on the *landing page*, and then click on the file to open it.

1.1.4 Rendering notebook as HTML

We'll use Quarto to print the `**ipynb*` file as HTML. Check the procedure for rendering a notebook as HTML [here](#). You have several options to format the file.

You will need to open the command prompt, navigate to the directory containing the file, and use the command: `quarto render filename.ipynb --to html`.

1.2 In-class exercise

1. Create a new notebook.
2. Save the file as `In_class_exercise_1`.
3. Give a heading to the file - `First HTML file`.
4. Print `Today is day 1 of my programming course`.
5. Compute and print the number of seconds in a day.

The HTML file should look like the picture below.

<IPython.core.display.Image object>

1.3 Python libraries

There are several [built-in functions](#) in python like `print()`, `abs()`, `max()`, `sum()` etc., which do not require importing any library. However, these functions will typically be insufficient for analyzing data. Some of the popular libraries in data science and their primary purposes are as follows:

1. NumPy: Performing numerical operations and efficiently storing numerical data.
2. Pandas: Reading, cleaning and manipulating data.
3. Matplotlib, Seaborn: Visualizing data.
4. SciPy: Performing scientific computing such as solving differential equations, optimization, statistical tests, etc.
5. Scikit-learn: Data pre-processing and machine learning, with a focus on prediction.
6. Statsmodels: Developing statistical models with a focus on inference

A library can be imported using the `import` keyword. For example, a NumPy library can be imported as:

```
import numpy as np
```


Using the `as` keyword, the NumPy library has been given the name `np`. All the functions and attributes of the library can be called using the `'np.'` prefix. For example, let us generate a sequence of whole numbers upto 10 using the NumPy function `arange()`:

```
np.arange(8)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

Generating random numbers is very useful in python for performing simulations (we'll see in later chapters). The library `random` is used to generate random numbers such as integers, real numbers based on different probability distributions, etc.

Below is an example of using the `randint()` function of the library for generating random numbers in `[a, b]`, where `a` and `b` are integers.

```
import random as rm
rm.randint(5,10) #This will generate a random number in [5,10]
```

7

1.4 Debugging and errors

Read sections 1.3 - 1.6 from http://openbookproject.net/thinkcs/python/english3e/way_of_the_program.html

1.5 Terms used in programming

Read section 1.11 from http://openbookproject.net/thinkcs/python/english3e/way_of_the_program.html

2 Variables, expressions and statements

Some of the commonly used objects in python are numbers - integer and float, strings and bool (true/false). The data type of the object can be identified using the in-built python function `type()`. For example, see the following objects and their types:

```
type(4)
```

int

```
type(4.4)
```

float

```
type('4')
```

str

```
type(True)
```

bool

2.0.1 Practice exercise 1

What is the datatype of the following objects?

1. 'This is False'
2. "This is a number"
3. 1000
4. 65.65
5. False

2.1 Constants and Variables

A constant is a value that cannot be changed. It may be a number, string or any other datatype. Below are some examples of printing a constant:

```
print(4)
```

4

```
print("This is a string and also a constant")
```

This is a string and also a constant

```
print(False)
```

False

A variable is an object whose value can be changed. For example, consider the object below:

```
x = 2
```

In the above code the variable `x` has been assigned a value of 2. However, the value of `x` can be changed:

```
x = 3  
print("x =", x)
```

x = 3

Thus, the object `x` in the above code is a variable that refers to a memory location storing the constant value of 3.

2.1.1 Variable names

There are a some rules for naming variables:

1. A variable name must start with a letter or underscore _
2. A variable name may consist of letters, numbers, and underscores only

For example, some of the valid variable names are `*salary`, `text10`, `__varname`. *Some of the invalid variable names are `salary% 10text`, `varname)*`*

3. Variable names are case-sensitive. For example, the variable `Varname` will be different from `varname`.
4. There are certain *reserved words* in python that have some meaning, and cannot be used as variable names. These reserved words are:

```
<IPython.core.display.Image object>
```

Best coding practice: Variables should be named such that they are informative of the value they are storing. For example, suppose we wish to compute the income tax a person has to pay based on their income and tax rate. Below are two ways of naming variables to do this computation:

```
income = 80000
tax_rate = 0.15
print("Income tax = ",income*tax_rate)
```

```
Income tax = 12000.0
```

```
a = 80000
b = 0.15
print("Income tax = ",a*b)
```

```
Income tax = 12000.0
```

The former code chunk is better than the latter one as it makes the code easy to read and understand.

2.1.2 Practice exercise 2

2.1.2.1 Variables or constants?

In the statements below, classify the objects as variables or constants?

1. `value = "name"`
2. `constant = 7`
3. `another_const = "variable"`
4. `True_False = True`

2.1.2.2 Valid variable names?

Which of the following variable names are valid?

1. `var.name`
2. `var9name`
3. `__varname`
4. `varname*`

2.2 Assignment statements

Values are assigned to variables with the assignment statement (`=`). An assignment statement may have a constant or an expression on the right hand side of the (`=`) sign, and a variable name on the left hand side.

For example, the code lines below are assignment statements

```
var = 2
var = var + 3
```

2.3 Expressions

The mathematical operations and their corresponding operators are as follows:

1. Exponent: `**`
2. Remainder: `%`
3. Multiplication: `*`
4. Division: `/`
5. Addition: `+`
6. Subtraction: `-`

The operators above are in decreasing order of precedence, i.e., an exponent will be evaluated before a remainder, a remainder will be evaluated before a multiplication, and so on.

For example, check the precedence of operators in the computation of the following expression:

```
2+3%4*2
```

8

In case an expression becomes too complicated, use of parenthesis may help clarify the precedence of operators. Parenthesis takes precedence over all the operators listed above. For example, in the expression below, the terms within parenthesis are evaluated first:

```
2+3%(4*2)
```

5

2.3.1 Practice exercise 3

Which of the following statements is an assignment statement:

1. `x = 5`
2. `print(x)`
3. `type(x)`
4. `x + 4`

What will be the result of the following expression:

```
1%2**3*2+1
```

2.4 Converting datatypes

Sometimes a value may have a datatype that is not suitable for using it. For example, consider the variable called *annual_income* in the code below:

```
annual_income = "80000"
```

Suppose we wish to divide *annual_income* by 12 to get the monthly income. We cannot use the variable *monthly_income* directly as its datatype is a string and not a number. Thus, numerical operations cannot be performed on the variable *annual_income*.

We'll need to convert *annual_income* to an integer. For that we will use the python's in-built `int()` function:

```
annual_income = int(annual_income)
monthly_income = annual_income/12
print("monthly income = ", monthly_income)
```

```
monthly income = 6666.666666666667
```

Similarly, datatypes can be converted from one type to another using in-built python functions as shown below:

```
#Converting integer to string
str(9)
```

```
'9'
```

```
#Converting string to float
float("4.5")
```

```
4.5
```

```
#Converting bool to integer
int(True)
```

1

Sometimes, conversion of a value may not be possible. For example, it is not possible to convert the variable *greeting* defined below to a number:

```
greeting = "hello"
```

However, in some cases, mathematical operators such as `+` and `*` can be applied on strings. The operator `+` concatenates multiple strings, while the operator `*` can be used to concatenate a string to itself multiple times:

```
"Hi" + " there!"
```

```
'Hi there!'
```

```
"5" + '3'
```

```
'53'
```

```
"5"*8
```

```
'55555555'
```

2.5 User input

Python's in-built `input()` function can be used to accept an input from the user. For example, suppose we wish the user to onput their age:

```
age = input("Enter your age:")
```

```
Enter your age:34
```

The entered value is stored in the variable `age` and can be used for computation.

2.5.1 Practice exercise 4

Ask the user to input their year of birth, and print their age.

2.6 Commenting code

The `#` symbol can be used to comment the code. Anything after the `#` sign is ignored by python. Commenting a code may have several purposes, such as:

- Describe what is going to happen in a sequence of code
- Document who wrote the code or other ancillary information
- Turn off a line of code - perhaps temporarily

For example, below is code with a comment to describe the purpose of the code:

```
#Computing number of hours of lecture in this course  
print("Total lecture hours of STAT201=",10*3*(5/6))
```

Total lecture hours of STAT201= 25.0

2.6.1 Practice exercise 5

Which of the following lines is a comment:

1. `#this is a comment`
2. `##this may be a comment`
3. `A comment#`

2.7 Programming errors

There are 3 types of errors that can occur in a program - syntax errors, run-time errors, and semantic errors.

2.7.1 Syntax errors

Syntax errors occur if the code is written in a way that it does not comply with the rules / standards / laws of the language (python in this case). For example, suppose a values is assigned to a variable as follows:

```
9value = 2
```

The above code when executed will indicate a syntax error as it violates the rule that a variable name must not start with a number.

2.7.2 Run-time errors

Run-time errors occur when a code is syntactically correct, but there are other issues with the code such as:

- Misspelled or incorrectly capitalized variable and function names
- Attempts to perform operations (such as math operations) on data of the wrong type (ex. attempting to subtract two variables that hold string values)
- Dividing by zero
- Attempts to use a type conversion function such as int on a value that can't be converted to an int

For example, suppose a number is multiplied as follows:

```
multiplication_result = x * 4
```

The above code is syntactically correct. However, it will generate an error as the variable `x` has not been defined as a number.

2.7.3 Semantic errors

Semantic errors occur when the code executes without an error being indicated by the compiler. However, it does not work as intended by the user. For example, consider the following code of multiplying the number 6 by 3:

```
x = '6'  
x * 3
```

```
'666'
```

If it was intended to multiply the number 6, then the variable `x` should have been defined as `x=6` so that `x` has a value of type `integer`. However, in the above code `6` is a `string` type value. When a `string` is multiplied by an integer, say n , it concatenates with itself n times.

2.7.4 Practice exercise 6

Suppose we wish to compute tax using the income and the tax rate. Identify the type of error from amongst syntax error, semantic error and run-time error in the following pieces of code.

```
income = 2000
tax = .08 * Income
print("tax on", income, "is:", tax)
```

```
income = 2000
tax = .08 x income
print("tax on", income, "is:", tax)
```

```
income = 2000
tax = .08 ** income
print("tax on", income, "is:", tax)
```

2.8 Practice exercise 7

The formula for computing final amount if one is earning compound interest is given by:

$$A = P \left(1 + \frac{r}{n} \right)^{nt},$$

where:

P = Principal amount (initial investment),

r = annual nominal interest rate,

n = number of times the interest is computed per year,

t = number of years

Write a Python program that assigns the principal amount of \$10000 to variable P , assign to n the value 12, and assign to r the interest rate of 8%. Then have the program prompt the user for the number of years t that the money will be compounded for. Calculate and print the final amount after t years.

What is the amount if the user enters t as 4 years?

A Datasets

Datasets used in the book can be found [here](#)