The Health Informaticist

Hants Williams

2024-10-28

Table of contents

Preface

Why?

I started off, and still identify as a nurse, but now find myself somewhere between the interaction of clinician and technologist. My job as a professor involves teaching, and constantly aquiring new knowledge, abilities, and skills (KSAs) between evidence based medicine (EBM) and technology-focused tools that could be used to help us address these issues. My curiosity and passion for BOTH healthcare and technology has led me to pursue a variety of topics that I try to address in this book. So what does this book cover?

This book covers python, healthcare data, medical codexes, open source datasets, databases, cloud technologies, inferential statistics and machine learning, visualizations, and related technologies important to understand as a modern (future) health informaticsts.

I've created this book as the most important topics to me, and what I have experienced within academic medical hospital systems, private hospitals, consulting, and health tech-startups. So whether you're a healthcare professional, data scientist, student, or enthusiast, this book will offer you valuable insights and hopefully fun conversation and dialgue related to what can be dry and boring material.

What You'll Find Here:

- **A Broad Foundation**: Rather than in-depth on each topic, this book provides a wide overview, giving you a base to explore further.
  - An intro to...*Python*
  - An intro to...*Healthcare Data*
  - An intro to...*Inferential Statistics*
  - An intro to...*Ai and Machine Learning*
  - An intro to...*Supporting Cloud Technologies*
- **Hands-On Learning**: Chapters combine theory and practical examples, allowing you to apply what you learn through Pyodide-powered, interactive Python exercises.

Think Legos...

Think of each section as a component, or a lego, that when combined together can lead to the creation of something useful. These core technologies: databases, scripts, clouds, code languages, databases, medical codexes...etc that we will be exploring should be viewed as individual lego pieces. It is our job as health informaticists to understand which pieces exist, what they are capability of providing to us, and then how we can put them together to make something unique and useful. Key word is useful.

Figure 1: Image of LEGO bricks as a metaphor for learning blocks

1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

2 Python Code - 1 - Hello World

2.1 Objectives

The aim of this exercise is to learn how to write and execute Python code.

2.2 Rationale

There are plenty of programing languages, but all of them share some common principles. Computers execute machine code, a binary set of instructions that are carried out by the computer processors, but usually human do not write binary machine code, or even assembly language, the written code directly translatable to machine code.

Most of the time programers write code in languages that are easier to write and read by us, humans. These codes, in turn, have to be translated to machine code. This process of translation is called compilation. Compilation is a complex process that comprises several steps, and different compilers and programming languages divide theses steps differently. There are compilers, like the C or C++ compiles, that take the code and compile it into a binary executable that can be directly run in a computer, while there are other, like the Python interpreter of the Java compiler, that are capable of running directly the human readable code that create an intermediate representation called bytecode. In these cases we need to install the language interpreter to run the program. For instance, if you want to run a Java or Python program you will need to install Java or Python before.

Python is an interpreted language, so, once you install Python you will be able to directly run Python code. However, you first need to have Python available in your computer.

There are many ways of installing Python like:

- Downloading Python from python.org.
- Using the Windows store or the Linux packages already prepared.
- Using a Python version manager like uv.
- Having Python included in your web browser thanks to Pyodide.

In this exercise we are going to use Pyodide, a Python version that allow us to run Python code directly in a web page without requiring any previous installation.

The Notebook document is composed by cells, some are text (in markdown format), and others contain Python code that can be executed. To execute a Python cell:

- Select the cell that you want to execute.
- Click the play symbol on top of the page or press shift + enter.

2.3 Print Hello world

```
#| exercise: hello_world_01
print("Hello world")
```

2.4 Print Hello "Your Name"

```
#| exercise: hello_world_02
# Use this Notebook cell to write Python code capable of printing your name
```

Solution.

> Tip
>
> ```
> print('Jane')
> ```

3 Python Code - 2 - Flow and Variables

3.1 Resources

- Variables in Python in Real Python.
- print function official documentation.
- A print function tutorial in Real Python.
- Variable unpaking tutorial in Real Python.

3.2 Flow

The computer executes the programming code one line (or statement) at a time. The order in which the lines are executed is called flow.

```
print("This line will be executed first.")
print("This line will be executed second.")
print("This line will be executed last.")
```

Order the lines in the following code, we want the computer to first say Hello, then Your Name, and, finally, an invitation to play.

```
#| exercise: fix_flow
print("Your Name")
print("Do you want to play a nice game of chess?")

print("Hello")
```

> **Note**
>
> Remember that the lines are executed in order, so change the order of the lines.

Solution.

Tip

```
print("Hello")
print("John")
print("Do you want to play a nice game of chess?")
```

3.3 Variables

When we run programs we store information in the memory of the computer for later use. This is a fundamental idea in computers. You can think about the memory of a computer as a file cabinet in which we can store values like numbers and strings of letters. Internally the memory is divided in small pieces, like the drawers of a filing cabinet, and the computer assings a numeric adress to each of those drawers. So, for instance, we could tell the computer to store the number 42 in the address 00000100, but it would be very cumbersome to use these numeric addresses, so, instead, the we use variables, names that we create, to refer to the memory locations and contents.

In different programming languages a variable can refer to the value stored or to the memory address (pointer). In Python a variable will always be a reference to the object stored. You can think of it as a label in the file cabinet drawer or an arrow that points to the drawer.

In Python we store a variable in memory by using the assigment operator =.

```
favorite_number = 42
print("My favorite number is ", favorite_number)
```

Python is doing quite a lot of things for us when we write "favorite_number = 42":

1. It reserves a space in the memory to be able to store the object that is going to create.
2. It creates the object 42.
3. It assigns the favorite_number variable as a reference to the created and stored object.

3.4 Create and print variables

Create two variables, one with your name, and another one with your surname and print them.

```
#| exercise: variable_name
# Fix this code, by providing a name and surname

name =
surname =

print("Hello ", name, " ", surname)
```

Solution.

Tip

```
name = "Jane"
surname = "Doe"
print("Hello ", name, " ", surname)
```

Write the name and year of release of any movie using two variables.

```
#| exercise: movie
# write here the code to print the name and year of a movie
```

Solution.

Tip

```
movie = "The Matrix"
year = 1999
print(movie, "was released in", year)
```

3.5 Change the value of a variable

The variables can be changed to refer to different values/objects. Fix the program to show your real name by changing the value of the variable.

```
#| exercise: variable_change
name = "John"
name =

print("My name is ", name)
```

Solution.

Tip

```
name = "John"
name = "Alice"

print("My name is ", name)
```

Given the following code, think about the expected output, what will the program print when you execute it?

```
name = "FALKEN"
name = "SNAPE"
print(name)
name = "FALKEN"
print(name)
```

3.6 Variables are not text strings

Fix the following code to print the correct name of your Pokemons.

```
#| exercise: variables_and_strings
my_pokemon = "Pikachu"
your_pokemon = "Ampharos"
print("My favorite pokemon is", "my_pokemon")
print("Your favorite pokemon is", "your_pokemon")
```

Note

Remember that variable names are not enclosed by quotes and that if you put something inside a quote Python will consider it a text string.

Solution.

Tip

```
my_pokemon = "Pikachu"
your_pokemon = "Ampharos"
print("My favorite pokemon is", my_pokemon)
print("Your favorite pokemon is", your_pokemon)
```

It is very important to understand the difference between variables and text:

- variable names are not enclosed by quotes.

```
a_variable = 42
print("a_variable")
print(a_variable)
```

Fix the following code:

```
#| exercise: fix_variables
a = 4
b = 7
c = "a" + "b"
print("c is seven:", c)
```

Solution.

Tip

```
a = 4
b = 7
c = a + b
print("c is seven:", c)
```

3.7 Assignment unpaking

Python allows for several variables to be set at the same time.

```
name, surname = "Ada", "Lovelace"
print(name, " ", surname, "was the first programmer")

temp1, temp2, temp3 = 36.5, 37.5, 39.5
print("The temperature has risen: ", temp1, " ", temp2, " ", temp3)
```

4 Python Code - 3 - Types

4.1 Resources

- Real Python tutorial: basic data types
- Official documentation: types and operations.
- Basic introduction to type and variables, str and booleans. (In Spanish).
- type function official documentation.
- Real Python introduction to numbers and bool.
- A discussion about the floating point arithmethic in Python.

4.2 Types

In a computer language variables have types. For instance, we have already used numbers and text strings.

```
#| setup: true
#| exercise: type01

# this is to avoid a bug in quarto-live, that,
# for some reason does not print the result of type
def type(obj):
    return obj.__class__.__name__
```

```
my_number = 42
text = "My favorite number is"
print(text, my_number)
```

We can ask for the type of variable (or object).

```
#| exercise: type01

number = 42
print('Type of the variable "number" is:', type(number))
print('Type of the string "42":', type("42"))
```

The type for the number 42 is int and for the text string "My favorite number is" is str. In most computer languages text is called string, or something similar, because, for the computer, a text is a string of characters.

4.3 Number types: int and float

In the previous example the type for the number was int (integer), but, in Python, and most other languages, we have another very important numeric type: float (floating point number).

```
#| excercise: type02

integer = 42  # type is int
floating_point = 42.0  # type is float
```

The main practical difference between the integer and float types is that the arithmetic for integers will be exact, but for float will be only approximate. This is not a Python quirk, but a general feature of the computers.

```
number = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
print(f'{number:.50f}')
```

Another practical limitation, besides the inherent error due to the approximation, of the floating point arithmetic is that you have to be careful when you compare floats.

```
first_one = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
second_one = 1.0
print("Are first and second one the same number?", first_one == second_one)

# But they are quite close
import math
print("Are first and second one close?", math.isclose(first_one, second_one))

# with integers we don't have this problem
first_ten = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
second_ten = 10
print("Are first and second ten the same number?", first_ten == second_ten)
```

4.4 Other types: bool and None

bool (boolean) is a special type that can only have two values: True or False.

```
i_am_learning = True
print(type(i_am_learning), i_am_learning)
i_am_not_here = False
print(type(i_am_not_here), i_am_not_here)
```

Another special type that you will find quite often in Python code is None. This type can only have one value: None. It is usally employed when you want to have a variable that still has no value, it signifies that it is empty.

```
#| setup: true
#| exercise: type03

# this is to avoid a bug in quarto-live, that,
# for some reason does not print the result of type
def type(obj):
    return obj.__class__.__name__
```

```
#| excercise: type03
result_to_be_calculated = None
print(type(result_to_be_calculated), result_to_be_calculated)
# Now we can calculate the result
result_to_be_calculated = 42
print(result_to_be_calculated)
```

4.5 Python is dynamic

In Python we don't need to specify the type of a variable before using it, and we can even change the type of object that the variable refers to. That is not the case in other languages, like C or Rust. In those static languages the compiler needs to know the type of the variable beforehand, and the type of a variable can not be changed.

```
#| setup: true
#| exercise: type04

# this is to avoid a bug in quarto-live, that,
# for some reason does not print the result of type
```

```
def type(obj):
    return obj.__class__.__name__
```

```
#| exercise: type04
my_number = None
print(type(my_number), my_number)
my_number = 42
print(type(my_number), my_number)
my_number = "42"
print(type(my_number), my_number)
my_number = 42.0
print(type(my_number), my_number)
```

Modern Python has the option of specifying the types. This is not a requirement, but you will see some type hints in Python code.

```
my_number: int = 42
my_number: float = 42.0
my_number: str = "42"
```

Be careful because Python does not enforce those type hints. If you wanted to use them for anything else than documentation you would need a type checker like mypy. But if you are starting in programming, just forget about this, the idea to remember is that the objects refered to by the variables have types.

4.6 Type casting

Type casting or type conversion is the action of changing one type into another. Python has functions to do this type casting.

```
my_number = 42
print(type(my_number), my_number)
my_number = str(my_number)
print(type(my_number), my_number)
my_number = float(my_number)
print(type(my_number), my_number)
my_number = int(my_number)
print(type(my_number), my_number)
```

We can also type cast to boolean.

```
print(bool(42))
print(bool(0))
print(bool("42"))
```

Find out which int and str values will be True and False when converted to boolean.

```
#| exercise: bool_type_casting
print(bool(value_to_test))
```

> **Note**
>
> Try with different int and str values, like 0, 1, 2, 3, “Hello”, ” “,” “.

Solution.

> **Tip**
>
> ```
> # Any int, but 0, will be True
> print("0", bool(0))
> print("1", bool(0))
> print("2", bool(2))
> print("3", bool(3))
> # Any str, but the empty str, will be True
> print('""', bool(""))
> print('"Hello"', bool("Hello"))
> print('" "', bool(" "))
> ```

Try to convert a boolean to integer.

```
#| exercise: bool_to_int

# what would be the result of converting the boolean values to integers?
```

Solution.

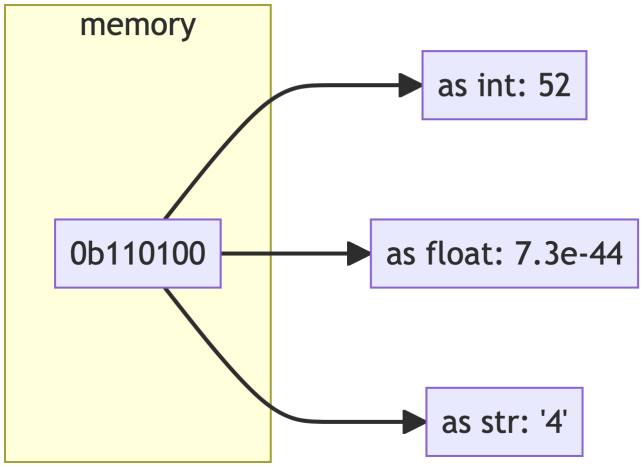
> **Tip**
>
> ```
> print("True", int(True))
> print("False", int(False))
> ```

4.7 Why do we need types?

For a programming language using types is unavoidable. Different languages have different types, but they all have types. Why? Because, internally, the only thing that a computer can really stores in memory is a binary number, and the same binary number can be interpreted as different objects depending on its type.

```
print("The same binary number can represented an integer or a string, depending on its type")
binary_number = 0b110100
print("The binary number is an integer that is usually printed in the decial number system:"
as_a_string = chr(binary_number)
print("The binary number as a string:", as_a_string)

print("The number 4 as an integer and as a string be stored by the computer in different ways
an_object = 4
print(f"The binary representation for {an_object} is: {an_object:b}")
another_object = "4"
print(f"The binary representation for {another_object} is: {ord(another_object):b}")
```



5 Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.