UNIVERSITY OF WATERLOO

Lesson 1.1: Prelude and Python Basics
Lessons Learned in CS 115

At Waterloo, many people colloquially call CS 115 "the **Racket** course" and CS 116 "the **Python** course." However, hopefully as graduates of CS 115, you know that this course is so much more than just learning **Racket**!

In CS 115, you learned about how to design programs. We introduced and developed a design recipe, a communication template, to help us convey to other programmers exactly what our code does and give evidence as to why it functions correctly. We discussed common design patterns, specifically using structural recursion to help us solve problems. Finally, yes, we *did* need to learn some **Racket** in the process! However, CS 115 was much more than just learning a new programming language!

Lessons to be Learned in CS 116

In CS 116, you will learn about **Python** yes but you will also learn so much more! We will maintain our design recipe but will need to modify it in the context of a new language. With a high level language like Python, our design recipe will have many more components that we will introduce in due time. Testing remains just as important as ever.

In this course, we also introduce a collection of new algorithms and discuss methods of analyzing which are most effective. We will keep our discussions in the language of Python but will try to introduce language features of Python that are common to many other languages as well so that students with a bit of effort should be able to convert the concepts learnt in Python to other programming languages. In this way, CS 116 is a course much more than just learning about Python!

By the end of this course, you should be able to do everything in Python that you could do in Racket but much more! This raises the question, *Why not learn Python to begin with?*

Functional vs. Imperative Programming Languages

Racket is a *functional* programming language. This means primarily that what you do in the language is create functions and evaluate mathematical expressions. Typically in this paradigm, states do not change and data is not mutable, that is, data cannot be changed. Think back to Racket when you defined a variable. Once set, that value remained constant for the duration of the program. Calculations are nested to show precedence and some calculated value is returned by the function upon termination. There are utilities and advantages of these languages that languages similar to Python do not have. As examples, testing tends to be easier, program verification is simpler, programs are easy to reason about and follow the program flow and so on.

To the contrary, Python is an *imperative* programming language. The biggest change is that data can be altered and states can change as the function executes. This allows us to execute far more complicated tasks, such as programming a GPS navigation system to take a driver from point A to point B. As you are driving, the state of where the car is changes and our GPS is updated according to display the correct information in real time. Usually, steps are ordered linearly (not nested like in a functional language) and calculated values may or may not be returned by functions (for example, our GPS system might choose to update the display screen and not return any value).

Concept Check 1.1.1

1 point possible (graded)
Fill in the blank. Python is ...

- ◯ a functional progrmaming language
- ◯ an imperative programming language
- ◯ an assembly programming language
- ◯ a visual programming language

Concept Check 1.1.2

1 point possible (graded)
What advantages do functional programming languages have over imperative languages? **Check all that apply.**

- ☐ They are easier to debug
- ☐ They are always faster
- ☐ They are simpler to read as they cannot change states of variables
- ☐ They are easier to trace

The Basics of Python

With imperative languages, the most common way to run a program is through a **compiler**. This compiler will consume the entire code as input and convert the entire code into machine code so that it can be run by your computer. In this way, the compiler can find all such errors and report them to the programmer.

Python however works more like Racket in this sense and uses an **interpreter**. This reads code one line at a time and stops if an error is found. This does make debugging easier but code that is compiled typically runs much faster.

Statements in Python

Python programs primarily consist of a sequence of three different types of statements:

1. Assignment statements
2. Control statements
3. Function calls

Later, we will also discuss statements such as function and class (type) definitions. However, the above three types of statements comprise most of what is executed by a Python program.

Assignment Statements

We begin our discussion in Python by discussing variable assignment. This is done using the **=** symbol. Formally, the syntax is of the form `v = expr`, where `v` is the variable name and `expr` is some sort of expression. For now, we will treat `expr` as a number but later we will see that this can, for example, be any arithmetic expression or more generally, any valid Python expression. This code will first evaluate `expr` and then assign the value of `v` to be this simplified expression.

Note that assignment statements do not return a value. They only have an effect, namely to assign a value to `v` above. For example, the following code sets the value of `a` to 4.3 and the value of `b` to 10.

```
1  a = 4.3
2  b = 10
```

Visualize | Reset Code

In the block above, try hitting the **Visualize** button! This will give us the first glimpse into our memory model.

To use the visualizer, click **Visualize** then click **Step Forward >**. This will cause the visualizer to execute the first line of code, namely `a = 4.3`. Pressing **Step Forward >** again will execute the second line of code `b = 10`. You can similarly use the **< Step Back** button to go in the reverse direction. You can also alter the code before you click visualize and the newly created code will be generated in the visualizer!

For now, variables will be treated as boxes where we place the value of the variable in the appropriately labelled box. In the above, after execution is complete, there should be a box in memory labelled `a` and the float `4.3` is in this box and there is a second box `b` with the integer `10` in it.

In Python, variable names are subject to a few constraints:

- A variable name must start with a lower or upper case letter or the underscore character. In this course, we will almost always start variables with lower case letters (with the exceptions of lists where it is common to use a single upper case letter).
- A variable name can only contain alpha-numeric characters (lower or upper case letters from a to z and digits 0 to 9) and underscores.
- Variables are case-sensitive, that is `var` and `Var` are two different variables.

Note in particular, this means that variables cannot begin with special characters or numbers. Notice further that dashes are not allowed in variable names which differs from Racket. If you ever want to put a hyphen in a name, use an underscore instead.

Concept Check 1.1.3

1 point possible (graded)
Fill in the blanks. Python uses a ********** which is unusual for ********** languages

- ◯ interpreter, imperative
- ◯ interpreter, functional
- ◯ compiler, functional

◯ compiler, imperative

Types

Python differs greatly from Racket in how it stores numbers. For Racket, all rational numbers were stored exactly. This included natural numbers and integers. So effectively, in Racket there is no difference between `3` and `3.0`.

Python however does not store rational numbers exactly! There is a huge difference with how Python stores the integer `3` and the rational number `3.0`. Integers are stored using a Python type called `int`, which we will denote by the data type `Int` in contracts. Contrastly, the rational number `3.0` is stored using a Python type called `float`, which we denote in contracts with `Float`. Notice in CS 116, we are retiring the data type `Num`, which we no longer use or reference.

The following table gives a clearer picture of how the data types from CS 115 compare to CS 116:

Data Type	Racket		Python	
	Representation	**Type**	**Representation**	**Type**
Natural	exact	`Nat`	exact	`Nat`
Integer	exact	`Int`	exact	`Int`
Rational	exact	`Num`	inexact	`Float`
Irrational	inexact	`Num`	inexact	`Float`

Lastly, please note that the official Python standard does not include `Nat` as a type but we will use it in our documentation for contracts.

There are other changes that we have in Python. Python doesn't have a `Sym` or `Char` data type. In lieu of the latter, we simply will use `Str` to use any string with 0 or more characters in it. More formally, a Python String is either:

- `""`, the empty string, or
- `"c" + t`, where `c` is any character and `t` is a string.

where above, `s + t` represents the new string when we concatenate the contents of `s` with the contents of `t`. Note that strings are not actually defined this way in Python but this definition will be useful for us.

Strings in Python can either be defined using single quotes, e.g., `'word'` or double quotes, e.g., `"another_word"`. This is useful for creating strings that might use one quote or the other. So, as an example, in the code below:

- The variable `s` contains the string `'Water'`
- The variable `t` contains the string `'loo'`
- The variable `w` contains the string `'Waterloo'`
- The variable `q` contains the string `"'"` (this is a string with a single quote)

- The variable `d` contains the string `'"'` (this is a string with a double quote)

```
s = "Water"
t = "loo"
w = s + t
q = "'"
d = '"'
```

Which of single or double quotes you use is a personal preference and can be swapped interchangeably (except in the instances where the string contains a quote, for example, `bakery_name = "Aria's Pies"`).

Concept Check 1.1.4

1 point possible (graded)
Which of the following gives a reasonable explanation as to why Python allows strings to be defined using two different types of quotes?

- ◯ Using multiple different kinds of quotes can improve readability.
- ◯ Using double quotes can allow for strings with single quotes and vice versa.
- ◯ This is a glitch in Python that we have to accept and work around it in certain situations.
- ◯ None of the above.

Concept Check 1.1.5

1 point possible (graded)
Which of the following are valid types for contracts in this course? **Check all that apply.**

- ☐ Nat
- ☐ Int
- ☐ Num
- ☐ Float

Concept Check 1.1.6

1 point possible (graded)
Consider the following code:

```
a = "12"
b = 2.0
```

What are the data types for `a` and `b`.

- ◯ Variable `a` is a `Str` and variable `b` is an `Int`.
- ◯ Variable `a` is a `Str` and variable `b` is a `Float`.

- ◯ Variable `a` is an `Int` and variable `b` is an `Int`.
- ◯ Variable `a` is an `Int` and variable `b` is a `Float`.
- ◯ None of the above

Final Thoughts

We introduced our new language Python in this lesson. We discussed variables and a memory model for how these are stored. These variables have a typing system that is a bit more complex than Racket. In particular, floating point values are stored inexactly in Python whereas in Racket, rational numbers were stored exactly. In the next sections, we will introduce Python arithmetic, discuss functions, and introduce new changes to the design recipe elements!