Explore Python syntax

Python is a flexible programming language used in a wide range of fields, including software development, machine learning, and data analysis. Python is one of the most popular programming languages for data professionals, so getting familiar with its fundamental syntax and semantics will be useful for your future career. In this reading, you will learn about Python's syntax and semantics, as well as where to find resources to further your learning.

The Language of Python

People use language to communicate and give instructions to each other. Computers do the same thing, except computers use languages like Python, C++, and Java. So, in order to communicate instructions to the computer, programmers need to arrange ideas and concepts into a language it will understand.

Python syntax includes words that represent objects and commands, as well as punctuation that gives the words structure, hierarchy, and context. Together, the words and punctuation communicate ideas and processes; this is known as semantics. Semantics is the meaning conveyed by the syntax. The best way to learn syntax and semantics is through exposure. Practice coding and become familiar and comfortable with reading other people's code. In addition, there are some general conventions that practitioners use to help maintain stylistic uniformity within the language.

Coding languages are similar to spoken languages in that they have a way to classify words according to their function. For example, English sentences are composed of nouns, verbs, prepositions, etc.

Here are some of the basics:

- **Variables:** Represent data stored as strings, tuples, dictionaries, lists, and objects (note: future readings explain these categories)
    - Example: `student_name`
- **Keywords:** Special words that are reserved for specific purposes and that can only be used for those purposes
    - Examples:
        - `in`
        - `not`
        - `or`
        - `for`
        - `while`
        - `return`
- **Operators:** Symbols that perform operations on objects and values
    - Examples:
        - `+` Addition
        - `-` Subtraction
        - `*` Multiplication
        - `/` Division
        - `**` Exponentiation

        - `%` Modulo (returns the remainder after a division). Example: `10 % 3` = `1`
        - `//` Floor division (divides the first operand by the second operand and rounds the result down to the nearest integer. Example: `5 // 2` = `2`
        - `>` Greater than (returns a Boolean of whether the left operand is greater than the right operand)
        - `<` Less than (returns a Boolean of whether the left operand is less than the right operand)
        - `==` Equality (returns a Boolean of whether the left operand is equal to the right operand)
- **Expressions:** A combination of numbers, symbols, and variables to compute and return a result upon evaluation
    - Example: `[1, 2, 3] + [2, 4, 6]`
- **Functions:** A group of related statements to perform a task and return a value
    - Example:

```
def to_celsius(x):
   '''Convert Fahrenheit to Celsius'''
   return (x-32) * 5/9

to_celsius(75)
```

RunReset

- **Conditional statements:** Sections of code that direct program execution based on specified conditions
    - Example:

```
number = -4

if number > 0:
   print('Number is positive.')
elif number == 0:
   print('Number is zero.')
else:
   print('Number is negative.')
```

RunReset

As you'll surely discover, Python generates syntax errors for incorrectly used keywords and syntax.

Example:

```
1
print(This will throw an error because I didn't make it a string.)
```

RunReset

Naming rules and conventions

When assigning names to objects, programmers adhere to a set of rules and conventions which help to standardize code and make it more accessible to everyone. Here are some naming rules and conventions that you should know:

- Names cannot contain spaces.
- Names may be a mixture of upper and lower case characters.
- Names can't start with a number but may contain numbers after the first character.
- Variable names and function names should be written in snake_case, which means that all letters are lowercase and words are separated using an underscore.
- Descriptive names are better than cryptic abbreviations because they help other programmers (and you) read and interpret your code. For example, student_name is better than sn. It may feel excessive when you write it, but when you return to your code you'll find it much easier to understand.

Tim Peters, a Python programmer, wrote this now-famous "poem" of guiding principles for coding in Python:

The Zen of Python

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one—and preferably only one—obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

Finally, it’s helpful to bookmark the PEP 8 Style Guide for Python so you can reference it as needed. This reading is limited in scope, and PEP 8 is a more exhaustive resource for style-related matters. PEP stands for Python Enhancement Proposals. These are a running catalog of ways to improve or standardize Python as a language. Because Python is open source, PEP offers a framework to guide developers and build consensus around ideas. It’s a useful and trusted resource.

Key takeaways

Syntax and semantics are what give form and meaning to a language, including Python. A large part of learning a new language is familiarizing yourself with its syntax and semantics. Much of this comes through exposure and practice, but there are a few guiding principles and resources that can help you along the way. If you learn the rules about naming objects and build a bank of resources that you can reference for guidance, you’ll surely make progress as a Python learner. As you get more familiar with Python, you’ll be able to communicate more efficiently with computers and do more with your data analysis tools!

Resources for more information

Here are a few useful resources to help you get more familiar with Python:

- Python Reference Library
  - Built-in Data types
  - Built-in functions
- Python operators