Data Scientist
Codecademy
Python Fundamentals
Python Syntax and Variable Types

Hello World

Welcome

Python is a programming language, which gives us a way to communicate ideas/commands with the computer!

We convey our commands to the computer by writing them in a text file using a programming language. These files are called programs. Running a program means telling a computer to read the text file, translate it to the set of operations that it understands, and perform those actions.

Comments

Text written in a program but not run by the computer is called a comment. Python interprets anything after a `#` as a comment.

Comments can:

- Provide context for why something is written the way it is.
- Help other people reading the code understand it faster.
- Ignore a line of code and see how a program will run without it.

Print

The gift of speech is valuable: a computer can answer many questions we have about "how" or "why" or "what" it is doing. In Python, the `print()` function is used to tell a computer talk. The message to be printed should be surrounded by quotes.

```
print("Type your message here")
```

The printed words that appear as a result of the `print()` function are referred to as output.

Strings

Computer programmers refer to blocks of text as strings. In Python a string is either surrounded by double quotes (“Hello World!”) or single quotes (‘Hello World!’). It doesn’t matter which kind we use, just be consistent.

Variables

Programming languages offer a method of storing data for reuse. Variables can store a value for later use. In Python, we assign variables by using the equals sign (=).

```
message_string = “Hello world”
print(message_string)
```

Variables cannot have spaces or symbols in their names other than an underscore (_). They cannot begin with numbers but they can have numbers after the first letter.

It is no coincidence we call them variables. If the context of a program changes, we can update a variable but perform a same logical process on it.

```
# Greeting
message_string = "Hello there"
print(message_string)

# Farewell
message_string = "Hasta la vista"
print(message_string)
```

Errors

Humans are prone to making mistakes. To compensate, programming languages attempt to understand and explain mistakes made in their programs.

Python refers to these mistakes as errors and will point to the location where an error occurred with a ^ character. When programs throw errors that we didn’t expect to encounter we call those errors bugs. Programmers call the process of updating the program so that it no longer produces unexpected errors debugging.

Two common errors that we encounter while writing Python are `SyntaxError` and `NameError`.

- `SyntaxError` means there is something wrong with the way our program is written – punctuation that does not belong, a command where it is not expected, or a missing parenthesis can all trigger a `SyntaxError`.
- A `NameError` occurs when the Python interpreter sees a word it does not recognize. Code that contains something that looks like a variable but was never defined will throw a `NameError`.

Numbers

Computers can understand much more than just strings of text. Python has a few numeric data types. It has multiple ways of storing numbers. The one we use depends on intended purpose for the number being stored.

An integer, or `int`, is a whole number. It has no decimal point and contains all counting numbers (1, 2, 3, ...) as well as their negative counterparts and the number 0.

A floating point number, or a `float`, is a decimal number. It can be used to represent fractional quantities as well as precise measurements.

Numbers can be assigned to variables or used literally in a program.

Floating point numbers can behave in some unexpected ways due to how computers store them.

Calculations

Computers absolutely excel at performing calculations. The “compute” in their name comes from their historical association with providing answers to mathematical questions. Python performs addition, subtraction, multiplication, and division with `+`, `-`, `*`, and `/`.

```
# Prints "500"
print(573 - 74 + 1)

# Prints "50"
print(25 * 2)

# Prints "2.0"
print(10 / 5)
```

Python converts all ints to floats before performing division.

Division can throw its own special error: `ZeroDivisionError`. Python raises this error when attempting to divide by 0.

Changing Numbers

Variables assigned numeric values can be treated the same as the numbers themselves. Two variables can be added or divided or multiplied by a third variable or literal without Python distinguishing between the variables and literals.

Performing arithmetic on variables does not change the variable, they can only be updated using the `=` sign.

```
coffee_price = 1.50
number_of_coffees = 4

# Prints "6.0"
print(coffee_price * number_of_coffees)
# Prints "1.5"
print(coffee_price)
# Prints "4"
print(number_of_coffees)

# Updating the price
coffee_price = 2.00

# Prints "8.0"
print(coffee_price * number_of_coffees)
# Prints "2.0"
print(coffee_price)
# Prints "4"
print(number_of_coffees)
```

Exponents

Python can also perform exponentiation. Since this operation is so related to multiplication, we use the notation `**`.

```
# 2 to the 10th power, or 1024
print(2 ** 10)

# 8 squared, or 64
print(8 ** 2)

# 9 * 9 * 9, 9 cubed, or 729
print(9 ** 3)

# We can even perform fractional exponents
# 4 to the half power, or 2
print(4 ** 0.5)
```

Modulo

Python offers a companion to the division operator called the modulo operator. The modulo operator is indicated by `%` and gives the remainder of a division calculation. If the number is divisible, then the result of the modulo operator will be 0.

```
# Prints 4 because 29 / 5 is 5 with a remainder of 4
print(29 % 5)

# Prints 2 because 32 / 3 is 10 with a remainder of 2
print(32 % 3)

# Modulo by 2 returns 0 for even numbers and 1 for odd numbers
# Prints 0
print(44 % 2)
```

The modulo operator is useful in programming when we want to perform an action every nth-time the code is run.

Concatenation

The `+` operator doesn't just add two numbers, it can also "add" two strings! The process of combining two strings is called string concatenation. Performing string concatenation creates a brand new string comprised of the first string's contents followed by second string's contents (without any added space in between).

```
full_text = greeting_text + " " + question_text

# Prints "Hey there! How are you doing?"
print(full_text)
```

To concatenate a string with number you will need to make the number a sting first, using the `str()` function.

In `print()` function we can pass numeric variable using commas to pass it as a argument rather than converting it to a string.

```
birthday_string = "I am "
age = 10
birthday_string_2 = " years old today!"

# Concatenating an integer with strings is possible if we turn the integer into a string first
full_birthday_string = birthday_string + str(age) + birthday_string_2

# Prints "I am 10 years old today!"
```

```
print(full_birthday_string)

# If we just want to print an integer
# we can pass a variable as an argument to
# print() regardless of whether
# it is a string.

# This also prints "I am 10 years old today!"
print(birthday_string, age, birthday_string_2)
```

Plus Equals

Python offers a shorthand for updating variables. When we have a number saved in a variable and want to add to the current value of the variable, we can use `+=` operator.

```
# First we have a variable with a number saved
number_of_miles_hiked = 12

# Then we need to update that variable
# Let's say we hike another two miles today
number_of_miles_hiked += 2

# The new value is the old value
# Plus the number after the plus-equals
print(number_of_miles_hiked)
# Prints 14
```

The plus equals operator can also be used for string concatenation.

```
hike_caption = "What an amazing time to walk through nature!"

# Almost forgot the hashtags!
hike_caption += " #nofilter"
hike_caption += " #blessed"
```

Multi-line Strings

Python strings are very flexible but if we try to create a string that occupies multiple lines we find ourselves face-to-face with a `SyntaxError`. Python offers a solution: multi-line strings

By using three quote marks (`"""` or `'''`) instead of one, we tell the program that the string doesn't end until the next triple-quote.

This method is useful if the string being defined contains a lot of quotation marks and we want to be sure we don't close it prematurely.

If a multi-line string isn't assigned a variable or used in an expression it is treated as a comment.