

Learn Python 3 : Syntax

Python `print()` Function

The `print()` function is used to output text, numbers, or other printable information to the console.

It takes one or more arguments and will output each of the arguments to the console separated by a space. If no arguments are provided, the `print()` function will output a blank line.

```
print("Hello World!")

print(100)

pi = 3.14159
print(pi)
```

Python Comments

A comment is a piece of text within a program that is *not* executed as part of the program. It can be used to provide additional information to aid in understanding the code.

The `#` character is used to start a comment and it continues until the end of the line.

```
# Comment on a single line

user = "JDoe" # End of line comment after code
```

Python Variables

A variable is used to store data that will be used by the program. This data can be a number, a string, a Boolean, a list or some other data type. Every variable has a name which can consist of letters, numbers, and the underscore character `_`.

The equal sign `=` is used to assign a value to a variable. After the initial assignment is made, the value of a variable can be updated to new values as needed.

```
# These are all valid variable names and assignment

user_name = "@sonnynomnom"
userID = 100
prompt21 = "Enter your user name"
existing_user = False
another_userID = userID

# A variable's value can be changed after assignment

score = 100
score = 120
```

Python Integers

Python variables can be assigned different types of data. One supported data type is the integer. An integer is a number which can be written without a fractional part (no decimal). An integer can be a positive number, a negative number or the number 0 so long as there is no decimal portion. The number `0` represents an integer value but the same number written as `0.0` would represent a floating point number.

```
# Example integer numbers

chairs = 4
tables = 1
broken_chairs = -2
sofas = 0

# Non-integer numbers

lights = 2.5
left_overs = 0.0
```

Floating Point Numbers

Python variables can be assigned different types of data. One supported data type is the floating point number. A floating point number is a value which contains a decimal portion. Floating point numbers are used to represent numbers which have fractional quantities. For example, `a = 3/5` can not be represented as an integer so the variable `a` is assigned the floating point value `0.6`.

```
# Floating point numbers
percentage = 0.75
pi = 3.14159
meal_cost = 8.49
```

Python Arithmetic Operations

Python supports different types of arithmetic operations. These operations can be performed on literal numbers, variables, or some combination. The primary arithmetic operators are:

- `+` for addition
- `-` for subtraction
- `/` for division
- `*` for multiplication

```
# Arithmetic operations

result = 100 + 300
result = 40 - 10
result = 50 * 5
result = 16 / 4

my_function(3 * 1.5)
```

Modulo Operator `%`

Python supports an operator to perform the modulo calculation. A modulo calculation returns the remainder of a division between the first and second number. For example:

- The result of the expression `4 % 2` would result in the value 0, because 4 is evenly divisible by 2 leaving no remainder.
- The result of the expression `7 % 3` would return 1, because 7 is NOT evenly divisible by 3, leaving a remainder of 1.

```
# Modulo operations

zero = 8 % 4

nonzero = 12 % 5
```

Exponentiation

In addition to the basic operations of addition, subtraction, multiplication and division, Python supports an operator for exponentiation. That operator is written with two asterisks like so `**`. The format for exponentiation in Python is a number or variable followed by the operator `**` followed by a number or variable which represents the power to raise the number. Both the number and the power can be integer or floating point values.

```
# Exponential operator

result1 = 2 ** 4

power = 2
square = 2 ** power

half = 0.5
result2 = square ** half
```

Python Integer Division

Python 3 will automatically convert integer numbers to floating-point before performing division. This behavior is changed from Python 2 where integer numbers were NOT automatically converted. In Python 2, dividing by an integer number performed integer division, where the fractional part (remainder) of the result is discarded. To perform regular division in Python 2, use of the `float()` function or a literal floating point value was required to force division to produce a floating point result.

```
# Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21)
result = 5 / 3
print(result)
# 1.6666666666666667

# Python 2.7.10 (default, Oct 6 2017, 22:29)
result = 5 / 3
print result
# 1
result = 5 / float(3)
print result
# 1.666666666667
```

Plus-Equals Operator `+=`

The plus-equals operator `+=` provides a convenient way to add a value to an existing variable and assign the new value back to the same variable. In the case where the variable and the value are strings, this operator performs string concatenation instead of addition. The operation is performed in-place, meaning that any other variable which points to the variable being updated will also be updated.

```
# Plus-Equal Operator

counter = 0
counter += 10

# This is equivalent to

counter = 0
counter = counter + 10

# The operator will also perform string concatenation

message = "Part 1 of message "
message += "Part 2 of message"
```

Python Strings

A string is a sequence of characters (letters, numbers, whitespace or punctuation) enclosed by quotation marks. It can be enclosed using either the double quotation mark `"` or the single quotation mark `'`.

If a string has to be broken into multiple lines, the backslash character `\` can be used to indicate that the string continues on the next line.

```
user = "User Full Name"
game = 'Monopoly'

longer = "This string is \
broken up \
over multiple lines"
```

Python String Concatenation

Python supports the joining (concatenation) of strings together using the `+` operator. The `+` operator is also used for mathematical addition operations. If the parameters passed to the `+` operator are strings, then concatenation will be performed. If the parameter passed to `+` have different types, then Python will report an error condition. Multiple variables or literal strings can be joined together using the `+` operator. The concatenation process does not add any whitespace between the strings that are joined.

```
# String concatenation

first = "Hello "
second = "World"

result = first + second

long_result = first + second + "!"
```

Error Notification

The Python interpreter will report errors present in your code. For most error cases, the interpreter will display the line of code where the error was detected and place a caret character `^` under the portion of the code where the error was detected.

```
if False ISNOTEQUAL True:
      ^
SyntaxError: invalid syntax
```

Python SyntaxError

A `SyntaxError` is reported by the Python interpreter when some portion of the code is incorrect. This can include misspelled keywords, missing or too many brackets or parenthesis, incorrect operators, missing or too many quotation marks, or other conditions.

```
age = 7 + 5 = 4

File "<stdin>", line 1
SyntaxError: can't assign to operator
```

Python NameError

A `NameError` is reported by the Python interpreter when it detects a variable that is unknown. This can occur when a variable is used before it has been assigned a value or if a variable name is spelled differently than the point at which it was defined. The Python interpreter will display the line of code where the `NameError` was detected and indicate which name it found that was not defined.

```
misspelled_variable_name

NameError: name 'misspelled_variable_name' is not defined
```

Python ZeroDivisionError

A `ZeroDivisionError` is reported by the Python interpreter when it detects a division operation is being performed and the denominator (bottom number) is 0. In mathematics, dividing a number by zero has no defined value, so Python treats this as an error condition and will report a `ZeroDivisionError` and display the line of code where the division occurred. This can also happen if a variable is used as the denominator and its value has been set to or changed to 0.

```
numerator = 100
denominator = 0
bad_results = numerator / denominator

ZeroDivisionError: division by zero
```