Full Stack Development
Sesi 1

Python Introduction

What is Python?

Python is a high-level, interpreted scripting language developed in the late 1980s by Guido van Rossum at the National Research Institute for Mathematics and Computer Science in the Netherlands. The initial version was published at the alt.sources newsgroup in 1991.

Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

Why Choose Python?

If you’re going to write programs, there are literally dozens of commonly used languages to choose from. Why choose Python? Here are some of the features that make Python an appealing choice:

- Python is Popular
- Python is Interpreted
- Python is Free
- Python is Simple

Integers

In Python 3, there is effectively no limit to how long an integer value can be. Of course, it is constrained by the amount of memory your system has, as are all things, but beyond that an integer can be as long as you need it to be:

```
print(123123123123123123123123123123123123123123123123 + 1)
```

Python interprets a sequence of decimal digits without any prefix to be a decimal number:

```
print(10)
```

```
print(type(10))
```

Floating-Point Numbers

Float values are specified with a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation:

```
print(4.2)

print(type(4.2))

print(4.)

print(.2)

print(.4e7)

print(4.2e-4)
```

Strings

Strings are sequences of character data. The string type in Python is called str.

String literals may be delimited using either single or double quotes. All the characters between the opening delimiter and matching closing delimiter are part of the string:

```
print("Hacktiv8")

print(type("Hacktiv8"))

print("This string contains a single quote (') character.")

print('This string contains a double quote (") character.')
```

Boolean

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

```
print(type(True))

print(type(False))
```

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
print(100 > 200) #False

print(100 == 200) #False

print(100 < 200) #True
```

Variable Assignment

To create a variable, you just assign it a value and then start using it. Assignment is done with a single equals sign (=):

```
n = 300

print(n)
```

Python also allows chained assignment, which makes it possible to assign the same value to several variables simultaneously:

```
a = b = c = 300

print(a, b, c)
```

Variable Names

Officially, variable names in Python can be any length and can consist of uppercase and lowercase letters (A-Z, a-z), digits (0-9), and the underscore character (_). An additional restriction is that, although a variable name can contain digits, the first character of a variable name cannot be a digit.

For example:

```
name = "Hacktiv8"

Age = 54

has_laptops = True

print(name, Age, has_laptops)
```

Variable Names

The most commonly used methods of constructing a multi-word variable name are the last three examples:

Camel Case, Example: numberOfCollegeGraduates

Pascal Case, Example: NumberOfCollegeGraduates

Snake Case,Example: number_of_college_graduates

Use whichever of the three is most visually appealing to you. Pick one and use it consistently.

Operators and Expressions

In Python, operators are special symbols that designate that some sort of computation should be performed. The values that an operator acts on are called operands. Here is an example:

```
a = 10

b = 20

print(a + b)

print(a + b – 5)
```

An operand can be either a literal value or a variable that references an object. A sequence of operands and operators, like a + b - 5, is called an expression. Python supports many operators for combining data objects into expressions.

Arithmetic Operators

```
# Here are some examples of these operators in use:

a = 4

b = 3

print(a + b)

print(a - b)

print(a * b)

print(a / b)

print(a % b)

print(a ** b)
```

Comparison Operators

Here are examples of the comparison operators in use:

```
a = 10

b = 20


print(a == b)

print(a != b)

print(a <= b)

print(a >= b)
```

String Manipulation

```
s = 'foo'

t = 'bar'

# + and * Operators

print(s + t )

print(s * 4)

#Case Conversion

print(s.capitalize())

print(s.lower())

print(s.swapcase())
```

Python Lists

Lists are defined in Python by enclosing a comma-separated sequence of objects in square brackets ([]), as shown below:

```
a = ['foo', 'bar', 'baz', 'qux']

print(a)
```

Characteristics of Python lists are as follows:

- Lists are ordered.
- Lists can contain any arbitrary objects.
- List elements can be accessed by index.
- Lists can be nested to arbitrary depth.
- Lists are mutable.
- Lists are dynamic.

Modifying List Value

A single value in a list can be replaced by indexing and simple assignment:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

a[2] = 10

a[-1] = 20

print(a)
```

What if you want to change several contiguous elements in a list at one time? Python allows this with slice assignment, which has the following syntax:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

print(a[1:4])

a[1:4] = [1.1, 2.2, 3.3, 4.4, 5.5]

print(a)
```

Python Tuples

Tuples are defined by enclosing the elements in parentheses (()) instead of square brackets ([]). Tuples are immutable. Here is a short example showing a tuple definition, indexing, and slicing:

```
t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')

print(t)

print(t[0])

print(t[-1])


# packing and unpacking

(s1, s2, s3, s4) = ('foo', 'bar', 'baz', 'qux')

print(s1)
```

Python Dictionary

Dictionaries and lists share the following characteristics:

- Both are mutable.
- Both are dynamic. They can grow and shrink as needed.
- Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

Dictionaries differ from lists primarily in how elements are accessed:

- List elements are accessed by their position in the list, via indexing.
- Dictionary elements are accessed via keys.

Defining and Accessing Dictionary

You can define a dictionary by enclosing a comma-separated list of key-value pairs in curly braces ({}). A colon (:) separates each key from its associated value.

```
MLB_team = {
    'Colorado': 'Rockies',
    'Boston': 'Red Sox',
    'Minnesota': 'Twins',
}
print(MLB_team['Minnesota'])
#Adding an entry to an existing dictionary
MLB_team['Kansas City'] = 'Royals'
MLB_team
```

Building a Dictionary Incrementally

You can start by creating an empty dictionary, which is specified by empty curly braces. Then you can add new keys and values one at a time:

```
person = {}

person['fname'] = 'Hack'

person['lname'] = '8'

person['age'] = 51

person['spouse'] = 'Edna'

person['children'] = ['Ralph', 'Betty', 'Joey']

person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}

print(person)

print(person['children'][1])
```

External References

Visit Here