Lesson 2: Variables

July 31, 2020

1 Lesson 2: Variables

In this lesson, we will learn about...

- Variable Assignment
- Primitive Data Types
- Type Casting

2 Variable Assignment

Assignment: Giving a variable a value

[1]:
```
a = 100
b = 1000.5
c = "Python"
```

2.1 Identifiers

- An Identifier is a name to identify a variable, function, class, module, etc.
- Starts with a letter or an underscore, followed by one or more letters, underscores, digits
- Class names start with a capital letter
- All other identifiers start with a lowercase letter
- Starting with an underscore is convention for a ”private” variable
- No punctuation allowed
- Case sensitive
- Separate words with underscores
- This is what Python’s Style Guide suggests for readability
- You may also see people using camelCase

2.2 Reserved Words

These are words with special meaning in Python, and cannot be used as identifiers. They usually change color in your IDE / editor.

[2]:
```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass
```

2.3 Multiple Assignment

Python allows you to assign values to multiple variables simulaneously.

[3]:
```
d = e = 1
f,g,h = 2,3,"Programming"
```

3 Data Types

Python has many built-in data types. We typically divide them into "primitives" and "non-primitives".

3.1 Primitive Data Types

These data types are the fundamental building blocks of a language. Python's primitives include...

- Text: `str`
- Numeric: `int`, `float`, `complex`
- Boolean: `bool`

We'll learn about non-primitive data types in a later lesson.

[4]:
```
this_is_a_string = "A String"
this_is_an_int = 5
this_is_a_float = 5.203
this_is_a_scientific_float = 1.5e6
this_is_complex = 5 + 3j
this_is_a_bool = True #(or False)

print(type(this_is_a_string))
print(type(this_is_an_int))
print(type(this_is_a_float))
```

```
print(type(this_is_a_scientific_float))
print(type(this_is_complex))
print(type(this_is_a_bool))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'float'>
<class 'complex'>
<class 'bool'>
```

- **Strings** are literal text, enclosed in single, double, or triple quotes.
- **Integers** are whole numbers, positive or negative (or zero), without decimals
- **Floats** are positive or negative (or zero) numbers with decimals. Floats can be expressed in scientific notation as well.
- **Complex** numbers have a real component and an imaginary component (represented with `j`)
- **Booleans**, named after mathematician George Bool, are binary values that are either `True` or `False`. The capitalization is important.

3.2 Type Casting

- Sometimes, you'll need to convert from one data type to another. This is called "type casting."
- Python will implicitly cast numbers when able to "larger" types to preserve data.
- Ex: `int` + `float` yields a `float`

[5]:
```
x = int(2.8)
y = float(3)
z = str(3)
w = int("7")
v = x + y

print(x)
print(type(x))
print(y)
print(type(y))
print(z)
print(type(z))
print(w)
print(type(w))
print(v)
print(type(v))
```

```
2
<class 'int'>
3.0
<class 'float'>
3
```

```
<class 'str'>
7
<class 'int'>
5.0
<class 'float'>
```

3.3 String Special Features

Strings represent a sequence of characters, and have some features we'll also see in the more complex data structures.

3.3.1 Get a Character

- Use brackets to get a character at a given position
- These **indexes** start with **the first character at 0**.

[6]:
```
hello = "Hello, World!"
print(hello[1])
```

```
e
```

3.3.2 Slicing / Substring

- Use the brackets with a colon to get a slice, or part, of the string from the start to end index (exclusive).
- `start` $\leq$ `end`
- **Tip:** `end` $-$ `start` $=$ length of slice
- You can also optionally leave out either the start or the end index

[7]:
```
hello = "Hello, World!"
print(hello[2:5])
print(hello[:5])
print(hello[9:])
```

```
llo
Hello
rld!
```

3.3.3 Negative Indexing

You can also use negative numbers to index from the end of the string! The last character is index `-1`.

[8]:
```
hello = "Hello, World!"
print(hello[-5:-2])
```

```
orl
```

3.3.4 Length

Use `len()` to get the length of, or number of characters in, a string.

[9]:
```
hello = "Hello, World!"
print(len(hello))
```

```
13
```

3.3.5 String Methods

There are various other methods you can run on a string. A full list can be found here.

Note: All string methods return a *new* string, and don't replace the original.

[10]:
```
hello = "Hello, World!"
print(hello.upper())
print(hello.lower())
print(hello.replace("H", "J"))
print(hello.split(","))
print("world" in hello)
```

```
HELLO, WORLD!
hello, world!
Jello, World!
['Hello', ' World!']
False
```

- `upper()` and `lower()` transform the string to all uppercase and lowercase characters respecitively
- `replace()` will replace all instances of the first string with the second, case sensitive
- `split()` will split the string into a list of substrings at whatever string is specified. That separator is not included in any substring.
- `x in y` will return `True` if `x` is a substring of `y`, and `False` otherwise. We'll learn more about boolean operators later

3.3.6 Concatenation

You can concatenate two strings by adding them.

[11]:
```
hello = "Hello, " + "World!"
print(hello)
```

```
Hello, World!
```

3.3.7 Escape Characters

Use escape sequences to insert invisible or otherwise illegal characters in strings. These include...

Escape	Result
\'	Single Quote
\"	Double Quote
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\\	Backslash

[12]:
```
hello = "He\"ll\"o,\tWor\nld\\!\b"
print(hello)
```

```
He"ll"o,    Wor
ld\
```