Learn To Program With Python 3

Data Types & Variables

Learn To Program With Python 3

Data Types

Data Types and Values

- Data: the quantities, characters, or symbols on which operations are performed by a computer

- Data Type: a **data type** or simply **type** is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.
  - Data types define the operations that can be done on that data.

- Value: The evaluated product of a singular piece of data.

Python Basic Data Types

- There are four “basic” types of data we will focus on.
  - Boolean – True/False (Research George Boole for background)
    - Example values: True, False
  - Integer – A whole number (not a fraction) that can be positive, negative, or 0. Do not use commas to separate the hundreds, thousands, etc places.
    - Example values: -1, 0, 1, 1028, 1238129127301012
  - Float – Represents high-precision fractional values (rational numbers).
    - Example values: 3.14, 1.5, 23.19283
  - String – A character is a letter of some alphabet, a digit, a blank space, a punctuation mark, etc. A String in python is sequence of one or more characters. Strings are usually used to represents words and text.
    - Example values: “a”, “1”, “1028”, “3.14”, “b”, “cat”, “dog”, “the quick brown fox jumped over the fence!”

- Remember “FIBS” – Float, Integer, Boolean, String

Why do data types matter?

- Data types define the operations that can be done on that data.
- *ALWAYS* think about **what data type** you are using, and **what operations** you can do with that data type!

```
# data_types_1.py
print(2 + 5) # you can add numbers
print('5' + '5') # you can add (concatenate) strings
print(2 + '5') # you CANNOT add strings and numbers!
```

```
7
55
Traceback (most recent call last):
  File "data_types_1.py", line 4, in <module>
    print(2 + '5') # you CANNOT add strings and numbers!
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

How to find the data type?

- You will not always immediately know what data type you are working with.
- You can use the built-in function known as type

class **type**(*object*)

class **type**(*name*, *bases*, *dict*)

With one argument, return the type of an ***object***. The return value is a type object and generally the same object as returned by `object.__class__`.

```
# data_types_2.py
# FIBS
print(type(3.14))
print(type(1028))
print(type(True))
print(type("Hello World!"))
```

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

ALWAYS

think about **what data type** you are using, and **what operations** you can do with that data type!

Exercises – Working With Strings

1. Read the documentation on strings here

Text Sequence Type — `str`

Textual data in Python is handled with `str` objects, or *strings*. Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways:

- Single quotes: `'allows embedded "double" quotes'`
- Double quotes: `"allows embedded 'single' quotes"`.
- Triple quoted: `'''Three single quotes'''`, `"""Three double quotes"""`

2. Read the documentation on string methods here

    1. There are many methods, only read a couple of them

`str.`**`capitalize`**`()`

Return a copy of the string with its first character capitalized and the rest lowercased.

Changed in version 3.8: The first character is now put into titlecase rather than uppercase. This means that characters like digraphs will only have their first letter capitalized, instead of the full character.

Exercises – Working With Strings

- Like a compiler (line by line, character by character), study the code to the right.
- Do you see anything new?
- What will it do?
- Try it!

```
# string_methods_1.py
print('Hello World!')
print('Hello World!'.islower())
print('Hello World!'.lower())
```

Exercises – Working With Strings

`str.`**`replace`**`(`*`old`*`, `*`new`*`[, `*`count`*`])`

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

```
# string_methods_2.py
greeting_one = 'Hello Juan!'
print(greeting_one)
greeting_two = greeting_one.replace('Juan', 'Erin')
print(greeting_two)
```

- Like a compiler, study the above.
- Do you see anything new?
- What will it do?
- Try it!

Exercises – Working With Strings

- Like a compiler, study the code to the right.
- Do you see anything new?
- What will it do?
- Try it!

```
# string_methods_3.py
age = 33
print(type(age))
print(age.lower())
```

Learn To Program With Python 3

Variables

Solve the following algebra problem:

X = 5

Y = 3

Solve: X + 2 + Y = ?

- If you can do the above problem, you already know what variables are!

Solve the following algebra problem:

- What does the following sentence mean?
- “ST 0900EST 4 121 IOT discuss DOD”
- We don’t know what these acronyms (variables) mean.
- We need to be told what they mean (they need to be defined) first.

Solve the following algebra problem:

- ST = “Show Time”
- EST = “Eastern Standard Time”
- 4 = “For”
- 121 = “One to One
- IOT = “In Order To”
- DOD = “Duties of the Day”

- Now read the sentence again: “ST 0900EST 4 121 IOT discuss DOD”
- Now that the acronyms (variables) have been defined, we can read the sentence!

Variables

- A variable is a placeholder for a value
- If it the compiler finds a variable that has not been defined, it throws an error.
- Once defined, you can use the variable name to refer to the value
- You can use the variable name to manipulate the value (change the value, update the value, etc)
- Variable names:
  - Can be arbitrarily long.
  - Must start with a letter but can contain letters and numbers.
  - Should be lowercase, with words separated by underscores as necessary to improve readability. This is known as “snake case”
  - this_is_a_good_snake_case_variable_name -> Snake case = good
  - Read more here

Variables

```
# variables_1.py
a_float_variable = 3.14
an_integer_variable = 1028
a_boolean_variable = True
a_string_variable = "Hello World!"

print(a_float_variable)
print(an_integer_variable)
print(a_boolean_variable)
print(a_string_variable)

some_random_variable = "The dog barks!"
print(some_random_variable)
some_random_variable = "Hi Juan!"
print(some_random_variable)
some_random_variable = 392
print(some_random_variable)
```

```
3.14
1028
True
Hello World!
The dog barks!
Hi Juan!
392
```

More on Variable Names

- There are certain variable names that the compiler will understand (starts with a letter, contains letters or numbers) but will not allow (throws syntax errors).
- For example: try creating a variable called “class”

```
# variables_2.py
class = "Learn to Program with Python 3"
```

```
  File "variables_2.py", line 2
    class = "Learn to Program with Python 3"
          ^
SyntaxError: invalid syntax
```

- “class” is a keyword (or “reserved word”) in python!

More on Variable Names

- Certain variables names are already taken by the Python programming language itself.
- These are known as reserved words, or keywords, and cannot be used as ordinary variable names. (source)

2.3.1. Keywords

The following identifiers are used as reserved words, or *keywords* of the language, and cannot be used as ordinary identifiers. They must be spelled exactly as written here:

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

Learn To Program With Python 3

Learn Full Stack Web Development

www.projectfullstack.com