Theory: Type casting

10 minutes 13 / 13 problems solved

Start practicing

§1. Dynamic vs. Static typing

Python is a **dynamically** and **strongly typed** language. Dynamic typing means that only runtime objects (values) have a type, but not the variables that store them. You are able to store several values of different types in a single variable during your code execution and no errors will occur.

The following code is totally valid:

```
v = 10  # variable v stores an integer value

v = 'hello'  # v now stores a string
```

On the other side, in **statically typed** languages each variable has a type that cannot be changed during the runtime, so the code above would fail. The examples of statically typed languages are C++, Java and Go.

§2. Strong vs. Weak typing

Strong typing means that implicit type conversions don't happen. For example, even though `"125"` consists only of digits it's a string. To use it in arithmetic operations you need to change its type to an integer or another numerical type. Trying to use it as is leads to a `TypeError`.

```
>>> "125" + 10
...
TypeError: can only concatenate str (not "int") to str
```

If Python had a **weak type system**, such value could be interpreted as an integer to successfully perform the operation. Such behavior, when one of the operands is implicitly converted to the type of another operand, is called **type coercion**.

Since there is no type of coercion in Python, the same operand may give different results depending on the types of provided arguments. For example, you can add two strings to get a concatenation. It is also possible to multiply a string by an integer:

```
print(125 + 125)  #  "250"

print("125" + "125")  # "125125"

print(125 * 4)  # "500"

print("125" * 4)  # "125125125125"

print("This is a number:", 125)  # "This is a number: 125"
```

The example also shows that you can print values of different types if you separate them with commas in the parentheses. The `print()` function will print all the arguments delimited by a space.

§3. Explicit type casting

The process of converting a value to another type is also called **type casting**. Though implicit type casting isn't allowed in Python you will often find yourself in need to define an explicit type conversion within your code. This happens a lot when you work with the user's input.

Imagine, you asked a user to provide an age that you will later use in some calculations. To convert a string to integer type you can use the built-in `int` function.

Current topic:

✓ Type casting 12★ ···

Topic depends on:

✓ Variables 17★ ··· Stage 1

Topic is required for:

Time module ···

XML in Python ···

Command line arguments ···

Intro to NumPy ···

Table of contents:

```
raw_age = "22"
print(type(raw_age))  # <class 'str'>


age = int(raw_age)
print(type(age))  # <class 'int'>
```

The `type` function is used to find out the type of the value provided.

To cast an integer to the string type use `str` function:

```
age = 22
print(type(age))  # <class 'int'>

string_age = str(age)
print(type(string_age))  # <class 'str'>
```

As you noticed, to cast a value to some type we use the function with the same name. If conversion between two types is not allowed you will see an error. Except for `str` and `int` functions we covered above there is also a `float` function. It converts a given value to the float type.

Here are some more examples of casting between different types:

```
f = 3.14  # the type is float
print(type(f))  # <class 'float'>

s = str(f)  # converting float to string
print(type(s))  # <class 'str'>


i = int(f)  # while converting a float value to an integer its fractional part is discarded
print(i)  # 3
print(type(i))  # <class 'int'>


f = float(i)

print(f)  # 3.0

print(type(f))  # <class 'float'>
```

It is important to remember that you can cast the value of any type to a string in Python. This fact is often used in debugging purposes.

Report a typo

1094 users liked this theory. **14** didn't like it. **What about you?**

Start practicing

Comments (18) Hints (0) Useful links (1) Show discussion