3. Data Types

Primitive Data Types - Storing Information

What are data types?

A program has to represent different kinds of values to work with.
Let’s start with the most basic ones: **primitives**.

Integer	Float	String	Boolean
0	0.89	“string”	True
1	1.7	“python”	False
2	2.0	“x”	
17593	-452.3	“DbPhfw”	
-452	…	…	
…			

Little Quiz – What's the Data Type?

`123`	Integer
`123.0`	Float
`treehouse`	Error! This is not a String! It's also no other valid data type!
`“89”`	String – Not an Integer
`“False”`	String – Not a Boolean
`true`	Error! Not a Boolean, and not a String! Not a valid data type!

Recap

- **Integer (int)**

  Numbers without decimals. Can be positive or negative.
- **Float (float)**

  Numbers with decimals. Can be positive or negative.
- **String (str)**

  Textual data, but really, can be any symbols as long as it's in quotation marks (“ ” or ‘ ’)
- **Boolean (bool)**

  Value that's returned by logical operations. Can only be **`True`** or **`False`** (watch out for case sensitivity!)

Variables

Let’s say, we have an Integer: 25. We want our computer to know about it.

We can’t just open up our code editor and enter 25. It won’t mean anything!

What do we do? Variables!

Maybe we wanted to represent in our program how old we are.
We write:

```
Variable = Data
```

Variables

For example:

```
alice = 25
```

Try it out with your own name and age!

Variables

Purpose of variables: **store data, reuse it, work with it.**

Let’s check if our program has stored our data. Try:

```
print(<yourvariable>)
```

LEGEND

- We use “<” and “>” to indicate **placeholders** in **code**.
- You need to **replace** the placeholders with reasonable values/text!

Variables

Let’s add another variable.

```
weather = “sunny”
```

Try to check if your program has stored this data.

Your code should look like this now:

```
<name> = <age>
print(<name>)
weather = “sunny”
print(weather)
```

Overwriting Variables

They’re not set in stone – we can modify them.

Let’s add a year to our age. We can simply assign a new value:

```
alice = 26

print(alice)

>>> 26
```

Overwriting Variables

We can also modify the data type of our variable. Let’s check its type now:

```
type(alice)

>>> <class ‘int’>
```

What if we assign alice a float value?

```
alice = 26.0

type(alice)

>>> <class ‘float’>
```

Overwriting: be careful!

Overwriting is easy, but because of this, it's also easy to mess something up.

```
age = 30

name = "Bob"

…

name = 31
```

Later in the code, we wanted to update the age, but accidentally modified the name.

Python won't throw an error here!
Other languages like Java or C++ are typed – for each variable, you predefine its data type. They wouldn't allow this. Python is more flexible.

Naming conventions

When naming your variables, there are things you cannot do.

For example: whitespaces

`alices age` is not a valid variable! Instead, e.g.: `alices_age`, `alicesAge`

In general:

- A variable name must start with a letter or the underscore character
- A variable name <u>cannot</u> start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords (if, else, import, return, …)