SIMPLE DATA TYPES

The simplest types of data you can store in a program are text strings and numerical data. These cards show you how to work with these simple data types.

VARIABLES

- What is a variable?
- What are the rules for naming a variable?

A *variable* is a label for a value you want to use in your program.

Variable names in Python follow two main rules:

- They must contain only letters, underscores, and numbers.
- They must start with a letter or underscore.

Good variable names are short but descriptive:

```
first_name = 'albert'
last_name = 'einstein'
username = 'einsteina'
age = 42
```

A variable can hold a small piece of information, or it can hold many gigabytes of data.

STRINGS

- What is a string?
- How do you store a string in a variable?
- How do you include tabs and newlines in a string?

A *string* is a value made up of one or more characters, surrounded by single or double quotes.

In this example, `"I love Python!"` is a string assigned to the variable `message`:

```
>>> message = "I love Python!"
>>> print(message)
I love Python!
```

Both single and double quotes work for strings, so you can use quotation marks inside a string:

```
>>> python_quote = 'I said, "I love Python!"'
>>> print(python_quote)
I said, "I love Python!"
```

Insert tabs and newlines into strings using the special sequences `\t` and `\n`:

```
>>> message = "Grocery list:\n\tmilk\n\teggs"
>>> print(message)
Grocery list:
    milk
    eggs
```

There is no limit on the length of a string.

STRING METHODS

- What is a string method?
- How do you change the case of a string?
- How do you strip whitespace from a string?

A *string method* is a function that performs an action on a string.

To change the case of a string, use the methods `title()`, `upper()`, and `lower()`:

```
>>> name = 'ella fitzgerald'
>>> name.title()
'Ella Fitzgerald'
>>> name.upper()
'ELLA FITZGERALD'
>>> name = 'Ella Fitzgerald'
>>> name.lower()
'ella fitzgerald'
```

The `lstrip()`, `rstrip()`, and `strip()` methods remove extra whitespace from strings, helpful for cleaning up data:

```
>>> name = '  jordan  '
>>> name.lstrip()
'jordan  '
>>> name.rstrip()
'  jordan'
>>> name.strip()
'jordan'
```

String methods are useful for presenting data in a certain format or cleaning up user-submitted data.

USING VARIABLES IN STRINGS

- How do you insert the value from a variable into a string?

In Python 3.6 onward, you can use a variable directly inside a string:

```
>>> username = 'efermi'
>>> print(f"Welcome back, {username}!")
Welcome back, efermi!
```

The `f` is short for *format* and tells Python to insert the value of the variable listed in braces inside the string. This tells Python to format the string using the given variable's value.

In Python 3.5 and earlier, you must use the `format()` method:

```
>>> username = 'efermi'
>>> msg = "Welcome back, {}!".format(username)
>>> print(msg)
Welcome back, efermi!
```

The placeholder {} will be replaced by the first variable included in the call to `format()`. You can include as many variables as you need:

```
>>> msg = "Welcome, {} and {}!".format(
      username_0, username_1)
```

COMMENTS

- How do you include a comment in your code?
- When should you include comments?

A *comment* is a line of text that's ignored by the Python interpreter. Comments allow you to leave notes for yourself or others inside a program.

Comments begin with a hash mark:

```
# Greet the user.
username = 'adrian'
print(f"Welcome back {username}!")
```

Comments are useful when writing complex code that you might need to return to. They can remind you of the reasoning behind the logic.

Clear, concise comments are a sign of a professional programmer. When considering different ways to solve a specific problem, include a comment that explains the approach you chose to take.

NUMERICAL DATA

- What are the two main types of numerical data?
- How do you find out what type of data you're working with?
- How do you convert between the two numerical types?

An *integer*, or *int*, is a number that doesn't have a decimal point. A *float* is a number that does.

The `type()` function identifies the data type of its argument.

```
>>> type(3)
<class 'int'>
>>> type(3.5)
<class 'float'>
```

The `float()` function converts an integer to a float:

```
>>> float(3)
3.0
```

Sometimes numerical information starts as a string. The `float()` function converts properly formatted strings to floats:

```
>>> float('3.5')
3.5
```

The `int()` function converts floats to integers by dropping the number's decimal portion, not by rounding:

```
>>> int(3.0)
3
>>> int(3.9)
3
```

The `int()` function also converts strings to integers:

```
>>> int('3')
3
```

NUMERICAL OPERATIONS

- How do you represent basic mathematical operations?
- How does the operation you're using affect the type of your output?
- How do you force Python to use a nonstandard order of operations?

Using addition, subtraction, or multiplication with integers returns an integer:

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
```

Using addition, subtraction, or multiplication with at least one float returns a float. All division operations return floats:

```
>>> 2.0 + 3
5.0
>>> 3.0 - 2
1.0
>>> 2.0 * 3
6.0
>>> 10 / 5
2.0
```

Two asterisks (**) represent an exponent. To raise 2 to the power of 3, enter:

```
>>> 2 ** 3
8
```

Use parentheses to force a nonstandard order of operations:

```
>>> (2 + 3) * 4
20
```

WORKING WITH NUMERICAL DATA

- How do you round numbers?
- How do you get the absolute value of a number?
- How do you convert a base-10 number to binary, octal, or hexadecimal?
- How do you represent complex numbers?

The `round()` function rounds a float to the given number of decimal places. Passing `round()` a negative argument results in multiples of 10:

```
>>> round(3.1415926, 2)
3.14
>>> round(1234, -2)
1200
```

The `abs()` function returns the absolute value of a number:

```
>>> abs(-5)
5
```

The `bin()`, `oct()`, and `hex()` functions convert base-10 numbers to base-2, base-8, and base-16 numbers:

```
>>> bin(20)
'0b10100'
>>> oct(20)
'0o24'
>>> hex(20)
'0x14'
```

The first two characters of the output indicate the new base.

The `complex()` function represents complex numbers:

```
>>> complex(2, 3)
(2+3j)
```

USING THE MATH LIBRARY

- How do you take the square root of a number?
- How do you use a mathematical constant such as pi?
- How do you use trigonometric and logarithmic functions?

The `math` library provides a number of functions for operations like taking square roots, using trigonometric functions, and working with mathematical constants:

```
>>> import math
>>> math.sqrt(9)
3.0
>>> PI = math.pi
>>> PI
3.141592653589793
>>> math.sin(PI/2)
1.0
>>> math.log(100, 10)
2.0
```

The standard library and third-party libraries also offer more mathematical functions. Before writing your own mathematical algorithm to solve a problem, be sure to look for an existing function or library.