

CSCI 145 · Introduction to Computing · Fall 2026

Modules & Strings

Lecture 4 — importing and using modules, string basics and functionality

Outline

- Importing from modules
-

- Strings
-

Part 1 of 2

Importing from modules

What if there is no built-in function for what you want to do?

Write your own function

as we did last week

Import a function from a module

that someone else has written

Using a module

At the top of your python file, you must **import** the module, using the **import** keyword, for example:

```
import math
```

Then, you can use functions in the module using `module_name.function_name` to indicate that the function you are calling is from the module, for example:

```
>>> math.sqrt(25)  
5.0
```

The **math** module

Function	Description	# of inputs
sqrt(x)	Returns the square root of x	1
pow(x, y)	Returns x to the power of y	2
sin(x)	Returns the sine of x	1
log(x, base)	Returns the log of x, in the given base	2
ceil(x)	Returns the smallest integer that is $\geq x$	1

None of these functions have side-effects; all return a value.

Trying `math` functions in the REPL

Function	Description
<code>sqrt(x)</code>	Returns the square root of x
<code>pow(x, y)</code>	Returns x to the power of y
<code>sin(x)</code>	Returns the sine of x
<code>log(x, base)</code>	Returns the log of x, in the given base
<code>ceil(x)</code>	Returns the smallest integer that is $\geq x$

```
>>> math.sqrt(16)
4.0
```

```
>>> math.pow(2, 10)
1024.0
```

```
>>> math.sin(0)
0.0
```

```
>>> math.log(100, 10)
2.0
```

```
>>> math.ceil(4.2)
5
```

The **random** module

Function	Description	# of inputs
random()	Returns a random float x such that $0 \leq x < 1$	0
randint(a, b)	Returns a random integer x such that $a \leq x \leq b$	2

None of these functions have side-effects; all return a value.

Trying **random** functions in the REPL

Function	Description
random()	Returns a random float x such that $0 \leq x < 1$
randint(a, b)	Returns a random integer x such that $a \leq x \leq b$

```
>>> random.random()  
0.6964691855978616
```

```
>>> random.randint(1, 6)  
4
```

Ways to import a module

Import statement	What it does	Calling a function
import math	same as <code>import math as math</code>	<code>math.sqrt(25)</code>
from random import *	import all functions	<code>randint(1, 6)</code>
from math import pow, sqrt	import specific functions	<code>sqrt(36)</code>
import turtle as t	can use <code>t.</code> as prefix	<code>t.forward(100)</code>

Learning about modules

- Use `dir` and `help` to learn about available methods, eg, `dir(math)` and `help(math.sqrt)`
- To find other modules that are available, check out python's documentation online: docs.python.org/3/library/index.html
- Some examples:
 - Math and more math: [math](#), [random](#), [fractions](#), [statistics](#)
 - Advanced ways to store data: [collections](#)
 - Accessing files and working with types of files: [os](#), [csv](#), [json](#)

Activity

1. Download [day04.zip](#).
2. Drag `day04.zip` to your `class` folder. Extract it there: **Mac** — double-click it. **Windows** — right-click it and choose "Extract All...". Once you see the `day04` folder, delete `day04.zip`.
3. Launch Thonny. From the File menu, open `cs145/class/day04/day04_modules.py`.
4. Rewrite part 1 of [Lab 1](#), using the `pow` and `sqrt` functions from the `math` module.

Part 2 of 2

Strings

Defining strings

- Strings are sequences of characters
- They are delimited by either single (') or double (") quotes

```
my_string = "Hello CS 145!"
```

Each character in `my_string` has an **index**:

my_string	H	e	l	l	o		C	S		1	4	5	!
index	0	1	2	3	4	5	6	7	8	9	10	11	12



Indices start from 0!

Accessing characters (indexing)

my_string	H	e	l	l	o		C	S		1	4	5	!
index	0	1	2	3	4	5	6	7	8	9	10	11	12

- You can access a single character by **indexing** with `[ ]`
- Indices range from 0 to the length of the string minus 1

```
>>> my_string[1]
'e'
```

```
>>> my_string[6]
'C'
```

```
>>> my_string[13]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Accessing a substring (slicing)

my_string	H	e	l	l	o		C	S		1	4	5	!
index	0	1	2	3	4	5	6	7	8	9	10	11	12

You can access multiple characters at once by **slicing** — a slice of a string is itself a string. A slice **excludes** the character at its ending index.

```
>>> my_string[1:4]
'ell'
```

```
>>> my_string[8:]
' 145!'
```

```
>>> my_string[:5]
'Hello'
```

Negative indexing

Each character also has a **negative index**, counting backward from the end.

negative index	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
my_string	H	e	l	l	o		C	S		1	4	5	!
index	0	1	2	3	4	5	6	7	8	9	10	11	12

We can access the `!` with either `my_string[12]` or `my_string[-1]`.

EXERCISE

What are some ways to access the substring `"145!"` — using positive or negative indices, but not both at once?

ANSWER

Positive indexing: `my_string[9:13]` or `my_string[9:]`

Negative indexing: `my_string[-4:]`

Slicing with a step

negative index	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
my_string	H	e	l	l	o		C	S		1	4	5	!
index	0	1	2	3	4	5	6	7	8	9	10	11	12

- A slice can take a third part, a **step**, written `[start:end:step]`
- Leaving out `start` or `end` means "from the beginning" or "to the end"

```
>>> my_string[0:5:2]
'Hlo'
```

```
>>> my_string[1::2]
'e! S15'
```

```
>>> my_string[::-1]
'!541 SC olleH'
```

A range that doesn't make sense gives you an **empty string** `" "`, not an error.

String operators & functions

str1 + str2

Concatenation — a new string with str1's characters, then str2's

string * n

Repetition — a new string with string repeated n times

x in string

Membership — True if x appears in string

len(string)

Length — the number of characters in string

String operators — examples

Operator	Description
str1 + str2	Concatenation: joins two strings
string * n	Repetition: repeats the string n times
x in string	Membership: True if x appears in string
len(string)	Length: number of characters in string

```
>>> "Hi" + "there"
'Hithere'
>>> "hi" * 3
'hihihi'
>>> "i" in "hi"
True
>>> len("hi")
2
```

EXERCISE

What if we wanted a space between "Hi" and "there"?

ANSWER

"Hi" + " " + "there" or "Hi " + "there"

What is the value of `s` after this code runs?

```
s = "abc"  
s = "d" * 2 + s  
s = s + "" * 2  
s = s + "q"
```

Go to **slido.com** and enter the code:

3674841

Or [join directly here](#)



What is the value of `s` after this code runs?

```
s = "abc"  
s = "d" * 2 + s  
s = s + " " * 2  
s = s + "q"
```

`s = "abc"`
→ `s = "ddabc"`
→ `s = "ddabc" # no change`
→ **`s = "ddabcq"`**

Converting to and from strings

You can convert between strings and numbers with `str()`, `int()`, and `float()`.

```
>>> str(42)
'42'
```

```
>>> int("42")
42
```

```
>>> float("3.14")
3.14
```

help()

`help()` gives us **documentation** on functions and modules, pulled from their **docstrings**.

```
>>> help(print)
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
        string inserted between values, default a space.
    end
        string appended after the last value, default a newline.
```

Keyword arguments and `print()`

Keyword arguments are named arguments with default values — leave one out and you get its default, or set it by name to override that default.

`print()` takes two: `sep` and `end`, which control how it separates and ends its output.

```
>>> print("a", "b", "c")  
a b c
```

```
>>> print("a", "b", "c", sep="-")  
a-b-c
```

```
>>> print("Loading", end="")  
>>> print("...")  
Loading...
```

As `help(print)` showed: `sep` defaults to a space, `end` to `\n`.

Special characters: `\n`

Some characters in a string are written with a backslash — a **special character**.

```
my_string = "Hello\nCS 145!"
```

← a two-character escape sequence, not two characters

`\n` represents a newline. If we print it:

```
>>> print(my_string)
Hello
CS 145!
```

Escaping special characters

`\` is the **escape character** in Python:

<code>\n</code>	Newline
<code>\t</code>	Tab
<code>\"</code> or <code>\'</code>	A quotation mark that doesn't end the string
<code>\\</code>	A literal backslash

```
>>> s = "Hello CS 145!\nHere comes a \t tab. And a quote \" and a \\ backslash"
>>> s
'Hello CS 145!\nHere comes a \t tab. And a quote " and a \\ backslash'
>>> print(s)
Hello CS 145!
Here comes a      tab. And a quote " and a \ backslash
```

Strings are immutable

Once a string is created, it **cannot be changed in place**.

```
>>> my_string[0] = 'h'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Operators and methods like `+`, `*`, and `.lower()` don't modify a string — they build and return a **new** one.

String methods: standardizing text

string.lower()

A new string, all
lowercase

string.upper()

A new string, all
uppercase

string.strip()

A new string,
leading/trailing
whitespace removed

```
>>> "Apples".lower()  
'apples'
```

```
>>> "Apples".upper()  
'APPLES'
```

```
>>> " apples ".strip()  
'apples'
```

String methods: search & transform

`string.find(sub)`

Index where `sub` first appears, or `-1`

`string.replace(old, new)`

New string with `old` replaced by `new`

`string.split(sep)`

Breaks the string apart at each `sep`

```
>>> "hi there".find("there")
3
```

```
>>> "hi there".replace("hi", "hey")
'hey there'
```

```
>>> "a,b,c".split(",")
['a', 'b', 'c']
```

[← a preview of lists!](#)

Building strings from values: f-strings

OPTIONAL

Useful but not required for this course.

Concatenating values into a string with `+` gets clunky and everything has to already be a string.

```
>>> name = "Anna"
>>> age = 10
>>> "Hello, " + name + "! You are " + str(age) + "."
'Hello, Anna! You are 10.'
```

An **f-string** is a string literal prefixed with `f` that lets you drop values in directly with `{ }`:

```
>>> f"Hello, {name}! You are {age}."
'Hello, Anna! You are 10.'
```

← the `f` prefix is what makes `{ }` work

Activity

1. Launch Thonny. From the File menu, open `cs145/class/day04/day04_strings.py`.
2. For part 1, predict the values of the string slices, then uncomment the `print` statements one at a time and check your answers.
3. For part 2, use string slicing to extract substrings for a username and a domain name from an email address.

To do

- Review this slide deck and the corresponding notes: [Notes 5](#)
- Finish [Homework 1](#), due this Thursday
- Prepare for [Lab 2](#) this Friday
- Quiz on Lectures 3–4 next class — review the [Practice materials](#) page

Next time

Lecture 5 — conditionals