UCLA Samueli School of Engineering

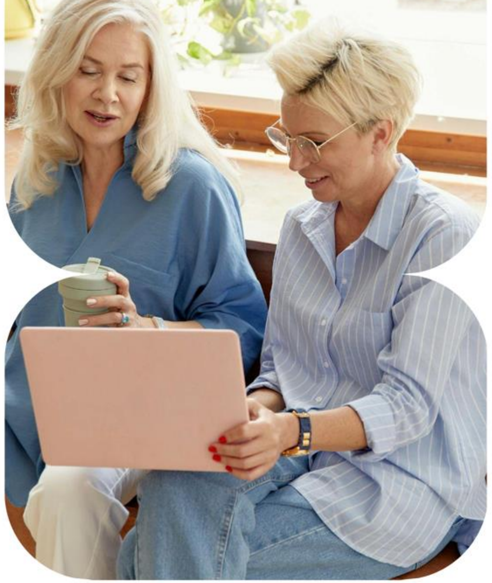
CS 35L Software Construction
Lecture 3 – Python

Tobias Dürschmid

Assistant Teaching Professor, Computer Science Department

Google

Fall 2025 Events @ UCLA



Come learn the ins and outs of preparing a resume so you can nail your ne
application! Google Software Engineers will pull back the curtains on the hiring process, typical timelines, and what to expect during interviews too.

Google Careers

Resume Workshop

Learn how to build a standout tech resume that gets noticed, with insider tips and best practices from Google. Food will be provided.

Building Your Technical Career

Join Googlers to learn how to build your technical skill and successfully navigate the path to tech internships and careers at Google. Food will be provided.

Thu Oct 16, 2025 @ 6-8 PM
Mong Learning Center
Engineering VI 180

Tue Oct 7, 2025 @ 5-7 PM
Mong Learning Center
Engineering VI 180

RSVP
bit.ly/UCLA-2025

careers.google.com/students

Course Logistics & Learning Tips

- **Slides & Code are posted to BruinLearn after exit tickets are due**
  - To encourage you to **recall lecture content** without looking back to previous slides (which has been shown to increase knowledge retention)
  - To encourage you to **actively participate in discussions** (which has been shown to improve your learning outcomes)
  - Aks questions when you are confused
  - Try to **think deeply about lecture content** rather than focus on taking notes (slides will be published, LAs publish their lecture notes)

Read more here: "*Make it stick: The science of successful learning*" by Henry L. Roediger III, Mark A. McDaniel, and Peter C. Brown.

Study Tips for Homework and Project

- Do the **tutorials** we recommend in the homework and slides
- Read `man` pages for all commands that you are using
  - If they are ambiguous:
    1. Find the Answer Online (Should be your first choice)
    2. Go test it yourself (works particularly well with less popular code or uncommon questions)
    3. Read the Source Code (the real pro move)
- Find a **study partner or study group** to discuss topics of this course. Learning together is more fun, and you get exposed to different ideas
  - **Make sure all submitted work is your own!**

Read more here: https://piazza.com/class/mg5edzlymak6fx/post/17

UCLA Samueli School of Engineering

Recap User Stories

CS 35 L SOFTWARE CONSTRUCTION

What is the INVEST Principle?

- **Independent**: Should be self-contained in a way that allows to be released without depending on one another.
- **Negotiable**: Only capture the essence of user's need, leaving room for design decisions.
- **Valuable**: Delivers value to end user.
- **Estimable**: Developers should be able to estimate the effort it takes to implement the user story
- **Small**: A user story is a small chunk of work that allows it to be completed in about 3 to 4 days.
- **Testable**: A user story has to be confirmed via pre-written acceptance criteria

Sidebar: Adversarial Thinking

- Look at your ideas from the **perspective of a critic** to identify weaknesses and violations of common design principles
  - Putting yourself in the shoes of a critic
  - Searching for vulnerabilities or weaknesses.
  - Actively challenging the assumptions of your ideas
- Apply adversarial thinking to the INVEST principle:
  - For each user story, try to find reasons why they might violate the INVEST principle
  - If you find weaknesses, improve your user stories
  - If even when trying very hard, you cannot find weaknesses, write down the reasons that convince the inner critic in yourself

Use Adversarial Thinking to Check Whether your User Stories Satisfy the INVEST principle

- **Independent**: Should be self-contained in a way that allows to be released without depending on one another.

Try to **find dependencies** between your user stories.
Does this user story A assume that some portion of user story B is already implemented?
If yes, try to **re-write** your user stories
If you cannot re-write them to be independent, **make the dependency explicit!**

Use Adversarial Thinking to Check Whether your User Stories Satisfy the INVEST principle

- **Negotiable**: Only capture the essence of user's need, leaving room for design decisions.

Try to find elements in your user story that are **design decisions** or that **over-constraint** the solution space (e.g., algorithms, technology, or other ideas on *how* to implement a feature)
If they are not really required by a user, remove them

Use Adversarial Thinking to Check Whether your User Stories Satisfy the INVEST principle

- **Valuable**: Delivers value to end user.

Does the user story as one increment to the software provide value on its own?
Try to think of what else might be needed to make the feature valuable? Are you missing something important? (e.g., only entering data without any means to retrieve data might be considered not valuable in many cases)

Use Adversarial Thinking to Check Whether your User Stories Satisfy the INVEST principle

- **Estimable**: Developers should be able to estimate the effort it takes to implement the user story

Try to find unintentionally vague wording that makes it ambiguous how much effort it would be to implement the user story (e.g., just saying the system should be support “some foreign languages” is not estimable since it is unclear how many and to what degree they should be supported

Use Adversarial Thinking to Check Whether your User Stories Satisfy the INVEST principle

- **Small**: A user story is a small chunk of work that allows it to be completed in about 3 to 4 days.

Try to break up your user story into smaller ones that are still "valuable". Repeat until you reached a point where smaller user stories are not "valuable" anymore.

Use Adversarial Thinking to Check Whether your User Stories Satisfy the INVEST principle

- **Testable**: A user story has to be confirmed via pre-written acceptance criteria

Try to think of unsatisfactory solutions to user story and ensure that the acceptance criteria rule them out.
Ensure that your acceptance criteria cover the most important functionality, critical corner cases, and common misconceptions.
Add or refine your acceptance criteria based on these insights.

UCLA Samueli School of Engineering

Recap Regular Expressions

CS 35 L SOFTWARE CONSTRUCTION

Write a Regular Expression to Find All Users

```
<h1>Bulk Download for Reviewers</h1>

As a reviewer (or administrator), I want to be able to download all submitted
files for a specific application in a single compressed folder (ZIP) so that I
can review the materials efficiently offline.

    Given I am logged in as a registered reviewer and viewing the details of a
specific submission,

    When I click the "Download All Submission Files" button,

    Then The system should generate and prompt me to download a single .zip
file containing every file uploaded by the applicant for that submission.

<h1>Applicant Registration</h1>

As an applicant, I want to be able to register and create a profile so that I
can easily track my submissions and receive official communications.

    Given I am a first-time user and I am on the system's registration page,

    When I fill in all required fields and click the "Register" button,

    Then My account should be created, and I should receive an email
confirmation with a link to my new profile dashboard.
```

Write a Regular Expression to Find All Users

```
<h1>Bulk Download for Reviewers</h1>

As a reviewer (or administrator), I want to
files for a specific application in a single compressed folder (ZIP) so that I
can review the materials efficiently offline.

    Given I am logged in as a registered reviewer and viewing the details of a
specific submission,

    When I

    Then T                                                        ip
file conta

<h1>Applic

As an appl                                                        t I
can easily track my submissions and receive official communications.

    Given I am a first-time user and I am on the system's registration page,

    When I fill in all required fields and click the "Register" button,

    Then My account should be created, and I should receive an email
confirmation with a link to my new profile dashboard.
```

Reg ex: As a .+,

We don’t want this, so we need an anchor so that we only match expressions that start the line with “As a”

Write a Regular Expression to Find All Users

```
<h1>Bulk Download for Reviewers</h1>

As a reviewer (or administrator), I want to
files for a specific application in a single compressed folder (ZIP) so that I
can review the materials efficiently offline.

    Given I am logged in as a registered reviewer and viewing the details of a
specific submission,

    When I click the "Download All Submission Files" button,

                                                                   ingle .zip
fi                                                                 mission.

<h1>Applicant Registration</h1>

As an applicant, I want to be able to register and create a profile so that I
can easily track my submissions and receive official communications.

    Given I am a first-time user and I am on the system's registration page,

    When I fill in all required fields and click the "Register" button,

    Then My account should be created, and I should receive an email
confirmation with a link to my new profile dashboard.
```

Reg ex: ^As a .+,

We miss users that start with a vowel

Write a Regular Expression to Find All Users

Reg ex: ^As an? .*,

```
<h1>Bulk Download for Reviewers</h1>

As a reviewer (or administrator), I want to
files for a specific application in a single compressed folder (ZIP) so that I
can review the materials efficiently offline.

    Given I am logged in as a registered reviewer and viewing the details of a
specific submission,

    When I click the "Download All Submission Files" button,

    Then The system should generate and prompt me to download a single .zip
file containing every file uploaded by the applicant for that submission.

<h1>Applicant Registration</h1>

As an applicant, I want to be able to register and create a profile so that I
can easily track my submissions and receive official communications.

    Given I am a first-time user and I am on the system's registration page,

    When I fill in all required fields and click the "Register" button,

    Then My account should be created, and I should receive an email
confirmation with a link to my new profile dashboard.
```

UCLA Samueli School of Engineering

Introduction to Python

CS 35 L SOFTWARE CONSTRUCTION

Python is an Interpreted Language

- Interpreted languages **do not have a compiler** (compared to C/C++)
- Code is executed line by line at runtime by an **interpreter**
- **Rapid prototyping** (no compiler step needed)
- Execution can be **slower** than compiled code (interpreter creates runtime overhead)
- **Requires Python to be installed on the target system**
  (in contrast to C/C++ which just compiles directly to the parget platform)
  - Software is delivered with the original source code

In-Class Discussion: What challenges might arise from writing systems in interpreted languages (compared to compiled languages)?

help – the man of Python

Terminal

```
$python
>>> 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.
    file: a file-like object (stream); defaults to the current sys.stdout.
    flush: whether to forcibly flush the stream.
```

When running the python command without specifying a script filename, the Python interpreter starts in **Interactive Mode.**

The * means you can pass multiple arguments to print separated by a

Hello World

simplehello.py

```
print("Hello_World!") #The simple print
```

multiple values

Separator symbol used when concatenating the list elements

end symbol displayed at the end of the concatenated list

hello.py

```
print("Hello", "World", sep='_', end='!\n')
```

Terminal

```
$python ./hello.py
Hello_World!
```

Python is a Dynamically Typed Language

- Variables can change their types dynamically

Terminal

```
>>> x = 4
>>> type(x)
<class 'int'>
>>> x = "4"
>>> type(x)
<class 'str'>
```

`type(var)` returns the class of which the variable is an instance of

In-Class Discussion: What challenges might arise from writing systems in a language with dynamic typing?

Python Arithmetic on Basic Types

Terminal

```
>>> "foo" + "bar"
'foobar'
>>>2 + 3
5
>>> 2.0 + 3
5.0
>>> 2.0 + "3.0"
TypeError: unsupported operand type(s) for +: 'float'
and 'str'
```

Data Types are Objects in Python

- **Basic Types:** int, float, bool (similar to C).
  - arithmetic operation with int and float —> float
- **Strings:** Strings are **immutable** (cannot be changed after creation) and a distinct type (str), not just an array of characters like in C
  - Build-in **sub string operator**: `string_example[start:end:stride]`
    - Default of start is 0
    - Default of stride is 1
  - Strings can be written either as `"string"` or as `'string'`
    - Allows you to add single or double quotation marks without escaping them as special characters e.g., `"my name is 'Tobias'"` or `'my name is "Tobias"'`

Substring Operations are Really Useful

Terminal

```
>>> "foobar"[:3] (same as "foobar"[0:3])
'foo'
>>> "foobar"[:-1]
'fooba'
>>> "foobar"[:-2]
'foob'
>>> "foobar"[::-1]
'raboof'
```

Negative end values count backwards from the last character

Reverses the string

f-Strings are Super Useful

- **f-Strings** (formatted strings) are a concise and efficient way to **embed Python expressions inside string literals for formatting**

Terminal

```
>>> print(f"5 * 11 = {5*11}.")
'5 * 11 = 55.'
>>> PI = 3.14159265
>>> print(f"Pi rounded is {PI:.2f}.")
'Pi rounded is 3.14'
>>> print(f"42 in binary is {42:b}.")
'42 in binary is 101010'
```

f-Strings start with `f""`

F-strings convert expressions inside braces to strings (calling the str operator)

Float with 2 decimals

Python has built-in List Data Structures

- Create a new list via: **`my_list=list()`** or **`new_list=[]`**
- Query the number of elements via: **`len(my_list)`**
- Add an element via: **`my_list.append(4)`** **or** **`my_list.insert(index, 4)`**
- You can remove an element via **`my_list.pop(index)`**
- Sort elements **that support an < operator** via **`my_list.sort()`**

Terminal

```
>>> new_list = []
>>> print(new_list)
[]
>>> new_list.append(3)
```

```
>>> print(new_list)
[3]
>>> print(len(new_list))
1
```

Defining Sequences in Python

- Sequences are defined via [expr(x) for x in sequence]

Terminal

```
>>> [x for x in range(1,10)]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [2*x for x in range(4)]
[0, 2, 4, 6]
>>> min([_ for _ in range(10)])
0
```

`range(start,end)` defines an integer sequence starting at `start` and ending at `end-1`

The default for `start` is 0

`min`, `max`, and `sum` are built-in functions

Python has great Syntactic Sugar

"Syntactic Sugar" refers to programming language features that make programming easier but are not adding any additional capabilities to the language

- You can unpack a list into variables: `[a, b] = [1, 2]`
- You can swap variables via: `a, b = b, a`

Terminal

```
>>> [a, b] = [1, 2]
>>> a
1
>>> b
2
```

```
>>> a, b = b, a
>>> a
2
>>> b
1
```

Python is an Object-Oriented Language

simplehello.py

```
class Book:
    """Represents a book with a title and an author."""

    def __init__(self, title, author, year):
        self.title = title
        self.author = author
        self.year = year


    def __str__(self):
        """
        Implements the str() operator (the user-friendly
        string representation).
        This is what print() calls.
        """
        return f'"{self.title}" by {self.author}
        ({self.year})'
```

Class comment

`__init__` is the constructor

Member variables are accessed via `self.varName`

Method comment

Incorrect indentation results in compiler errors!

```
# Create an instance of the class
my_book = Book("Pride and Prejudice",
"Jane Austen", 1813)


# 1. Using print()
print(my_book)
# Output: "Pride and Prejudice" by Jane
Austen (1813)


# 2. Using the str() function explicitly
print(str(my_book))

# 3. Using in an f-string explicitly
print(f"The book is: {my_book}")
```

Create a Book object

Calls `mybook.__str__()`

Equivalent to `print(my_book)`

Calls `mybook.__str__()`

Let’s use Python to read user stories via RegEx

regex.py

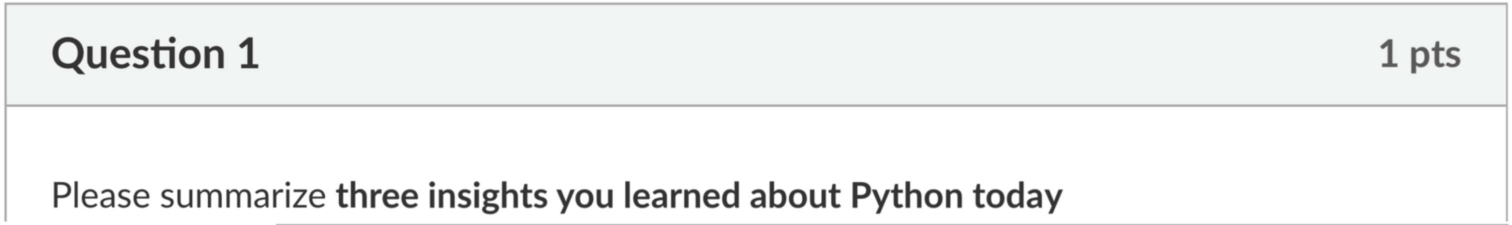
```
import re

import os

FILE_PATH = 'L3_file_example.html'


What next???
```

Imports the module with name ”re” (regex)

Convention to name constants in all upper case

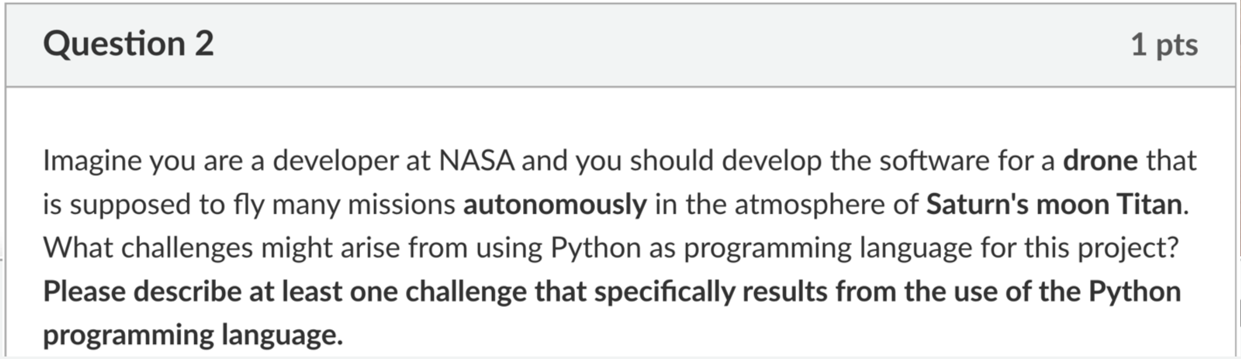
Check out the documentation at https://docs.python.org/3/library/re.html

Please fill out your Exit Tickets on Bruin Learn!



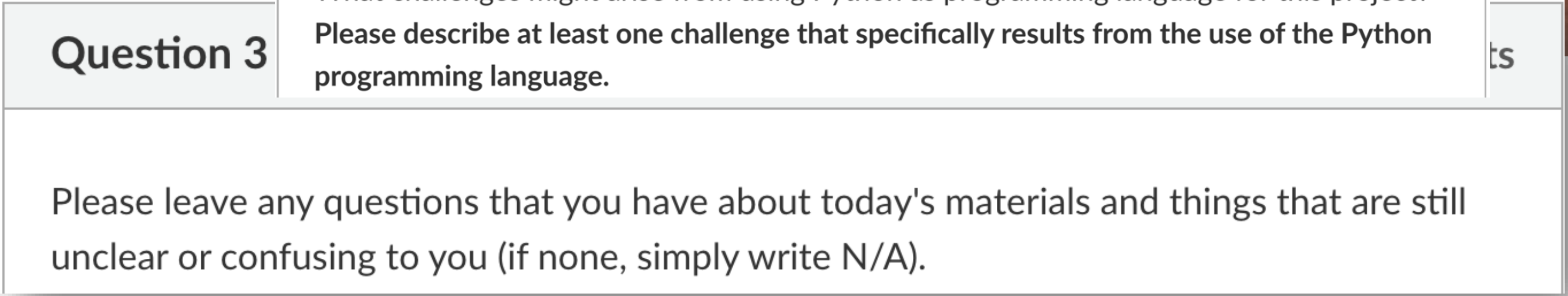
Question 1 1 pts

Please summarize **three insights you learned about Python today**



Question 2 1 pts

Imagine you are a developer at NASA and you should develop the software for a **drone** that is supposed to fly many missions **autonomously** in the atmosphere of **Saturn's moon Titan**. What challenges might arise from using Python as programming language for this project? **Please describe at least one challenge that specifically results from the use of the Python programming language.**



Question 3 ts

Please leave any questions that you have about today's materials and things that are still unclear or confusing to you (if none, simply write N/A).