RICE SOCIAL SCIENCES
Department of Linguistics

python essentials: types, strings, lists

LING 430 Computational Linguistics Fall 2026
lecture 2a
Sep 1 2026

Siyu Liang
siyu.liang@rice.edu

last time

- functions: `print`, `len`, `type`, `split`, `set`
- variables and assignment

agenda

- variables & types
- numbers
- strings
- lists

today's notebook

- course website → week 2 → opens in Colab
- **File → Save a copy in Drive**

variables & types

variable

- using an **alphanumeric name** to stand in for a **value**
- value could be a number, a string, a list, …

```
num = 17
num_add = num + 4
num_add
```

→ 21

- reassigning a name discards the old value

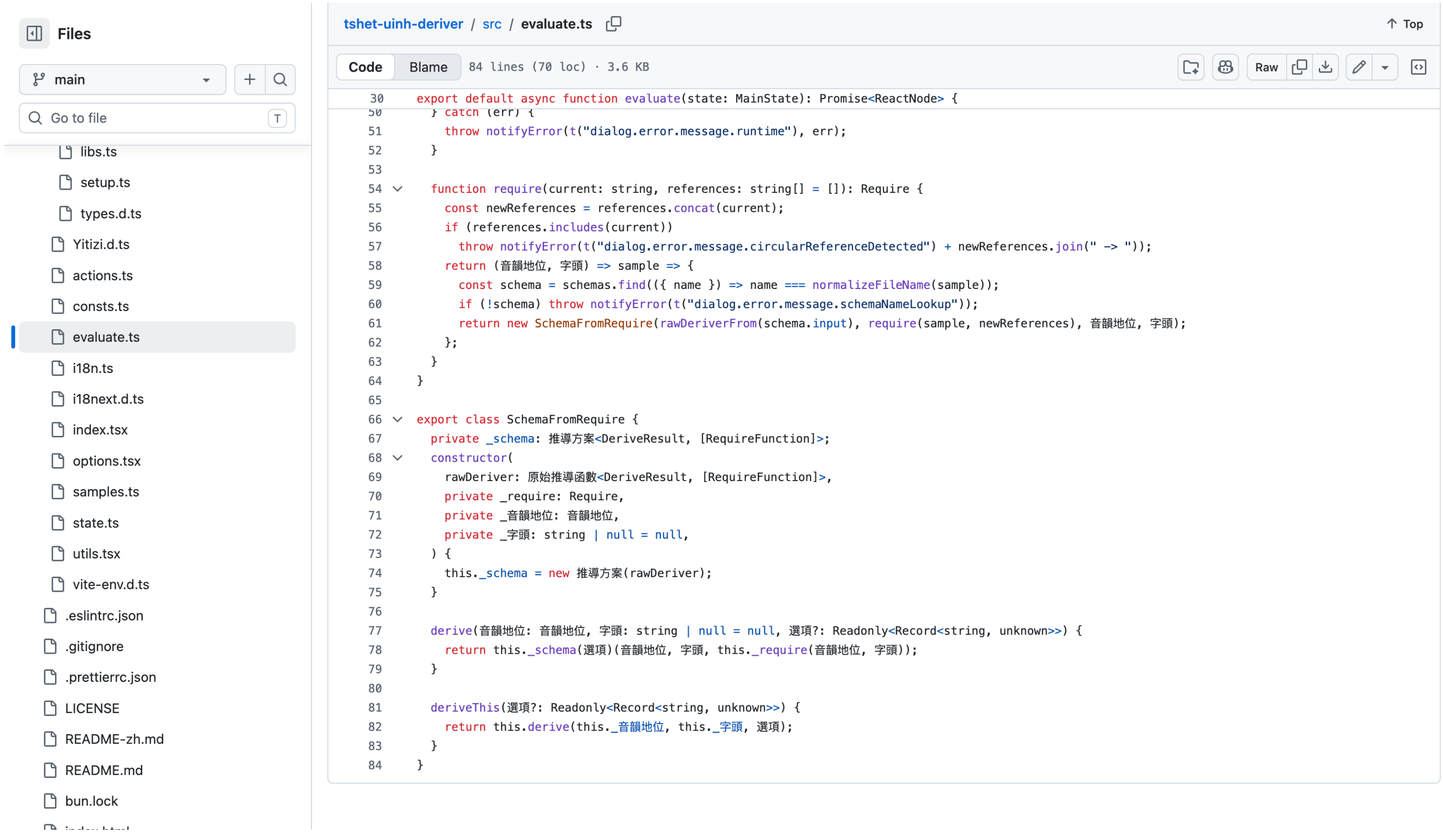
```
num = 3
num
```

→ 3

variable

- variable names don’t have to be in English

```
あ = "a"
```
OK



naming things well

- variables should be **informative**
  - `words`, `counts`, `stem`
- bad choices confuse you later
  - `x`, `data2`, `final_final`
- name for the reader
  - …who is usually future you

type

- tells you **what kind of thing** something is

  `type("Odyssey")`

  → str (a string)

  `type(7)`

  → int (an integer)
- quotes matter!
  - `"7"` is text, `7` is a number

type

- types are determined **automatically** in Python

```
x = 17
type(x)
```

→ int

```
x = "seventeen"
type(x)
```

→ str

type

- each type's name doubles as a converter function
- can be helpful to fix `TypeError`s such as in `"rose" + 3`

  `float(3)`

  → 3.0

  `int(42.1)`

  → 42

  `str(42)`

  → '42'

  `round(2/3, 2)`

  → 0.67

type

recap

Object type	Example creation
Numbers (int, float)	`123, 3.14`
Strings	`"this class is cool"`
Lists (incl. nested)	`[1, 2, [1, 2]]`
Dictionaries	`{"1": "abc", "2": "def"}`
Tuples	`(1, "Test", 2)`
Files	`open("file.txt"), open("file.txt", "w")`
Sets	`set('a', 'b', 'c')`
Others	`boolean, None`
Program unit types	`functions, modules, classes`

expressions vs statements

- an *expression* has a value

  `len(word), "rose" * 3`
- a *statement* does something

  `word = "hello"`
- a cell shows the value of its last **expression**
  - which is why some cells print without `print()`

numbers

int & float

- `int`
  - whole and exact, e.g., counts

    `type(42)`

    → int
- `float`
  - decimal, e.g., probability

    `type(42.0)`

    → float

- division always returns a `float`, even when it comes out even
- mostly because you want to be consistent

  `84 / 2`

  → 42.0

operator

- symbols or keywords that perform a specific action on values

  `6 * 7`

  → 42

  `84 / 2`

  → 42.0

  `2 ** 10`

  → 1024

  `count == 2`

  → True

operator

- floor

  ```
  7 // 4
  ```

  → 1

- modulus

  ```
  7 % 4
  ```

  → 3

  ```
  8 % 4
  ```

  → 0

comparisons

- you can use operators to compare variables
- the answers will validate to `True` or `False`, of the type *bool* (boolean)

  `7 > 4`

  → True

  `len("Odyssey") == 7`

  → True

  `"rose" != "Rose"`

  → True
- and others: `>`, `<`, `>=`, `<=`, `==`, `!=`

strings

quotes

- **single** and **double** quotes are **interchangeable**
  - choose whichever avoids escaping

```
s1 = "a string in double quotes contains 'single quotes' happily"
```

```
s2 = 'and vice "versa"'
```

- **triple** quotes hold **multi-line** text
  - useful for pasting in a passage

```
s3 = """triple quotes
may span lines"""
```

special characters

- `\n` is a newline

  `print("line one\nline two")`

  → line one

  → line two
- `\t` is a tab

  `print("word\tgloss")`

  → word gloss
- real language data and files are full of them
  - lines of a transcript, columns of a glossary
- tsv (tab-separated values) files

string operations recap

- concatenation

  `"un" + "happy”`

  → 'unhappy'
- repetition

  `"ha" * 3`

  → 'hahaha'
- length

  `len("unhappy")`

  → 7
- membership

  `"ppy" in "unhappy”`

  → True

indexing

- every character has a position in a string
- positions count from **0**; negative positions count from the end
- `word = "duckling"`
- `  d  u  c  k  l  i  n  g`
- `  0  1  2  3  4  5  6  7`
- `word[0]`
  - → 'd'
- `word[-1]`
  - → 'g'

slicing

- slicing lets you **select a range of positions**
- `[start:stop]`: from `start` up to but **not including** `stop`
  - `word[0:4]`
    - → 'duck'
  - `word[4:8]`
    - → 'ling'
- why not-including? so the two halves reassemble with no overlap
  - `word[0:4] + word[4:8]`
  - → 'duckling'

slicing

- **blank start**: from the beginning
  - `word[:4]`
    - → 'duck'
- **blank stop**: to the end
  - `word[4:]`
    - → 'ling'
- `[:-1]`: everything but the last character
  - `word[:-1]`
    - → 'ducklin'

slicing

- the full picture: `[start:end:step]`
  - `word[::-1]`
    - → 'gnilkcud'

mutable vs. immutable

- a string is **immutable**
- it can never be changed in place
- operations build new strings

```
s = "rice"
s[0] = "n"
```

→ TypeError: 'str' object does not support item assignment

```
s = "n" + s[1:]     # build a new string instead
s
```

→ 'nice'

- the same reason `s.lower()` without `s = s.lower()` changes nothing

strings

recap

```
s1 = 'the'
```

Operation	Description	Output
`len(s1)`	length of the string	`3`
`s1[0]`	indexing, 0-based	`'t'`
`s1[-1]`	backwards indexing	`'e'`
`s1[0:3]`	slicing, extracts a substring	`'the'`
`s1[:2]`	slicing, extracts a substring	`'th'`
`s1 + ' sun'`	concatenation	`'the sun'`
`s1 * 3`	repetition	`'thethethe'`

strings

recap

```
a = 'Anna'
```

Operation	Output
`a[::-1]`	`annA`
`'@'.join(a)`	`'A@n@n@a'`
`a[0:1]+"dmin"`	`'Admin'`
`a.find("a")`	`3`
`a.split()`	`['Anna']`
`"b@org.com".split("@")`	`['b', 'org.com']`

linguistic reduplication

- many languages build words by **copying part of the word onto itself**
  - Tagalog: *bili* ‘buy’ → *bibili* ‘will buy’
  - Indonesian: *rumah* ‘house’ → *rumah-rumah* ‘houses’
  - English has a contrastive version: is it a “salad-salad”? (or just cold pasta)

activity

Classical Greek perfect

present		perfect	
luō	“I release”	*leluka*	“I have released”
thuō	“I sacrifice”	*tethuka*	“I have sacrificed”

- write code that builds the perfect from the present form
- tips
  - input: present form >> need to get rid of the ‘o’, `word_without_o = ...`?
  - output:
    - need to duplicate first consonant: `first_consonant = ...`?
    - add the -ka suffix: `print(x+y+"ka")`

list

list

- a *list* is a collection of arbitrarily typed objects
- positionally ordered
- no fixed size
- unlike strings, **mutable**

```
myList = [123, 'spam', 1.23]
```

```
birds = ["duck", "goose", "swan"]
```

```
empty = []
```

list

strings vs. lists

- `.split()` turns one string into a list of its space-separated pieces

  ```
  line = "colorless green ideas sleep furiously"
  tokens = line.split()
  type(tokens)
  ```

  → list
- `len`, `in`, `.count()` all transfer
  - counting items, not characters

    ```
    len(tokens)
    ```

    → 5

list

- slicing works similarly too

  `tokens[0]`

  → 'colorless'

  `tokens[-1]`

  → 'furiously'
- a slice of a list is a smaller list

  `tokens[1:3]`

  → ['green', 'ideas']

list

- lists can contain lists
  - read the brackets left to right: row, then column

```
glosses = [["inu", "dog"], ["neko", "cat"], ["tori", "bird"]]
glosses[1]
```

→ ['neko', 'cat']

```
glosses[1][0]
```

→ 'neko'

list

- lists are **mutable**
- items in a list can be changed **in place**

```
x = ['a', 'b', 'c']
x.append('d')
x[2] = 'e'
x
```

→ ['a', 'b', 'e', 'd']

list

- `=` does not copy a list, it adds a second name for the **same** list

  ```
  a = ["ba"]
  b = a
  a.append("baa")
  b
  ```

  → ['ba', 'baa']
- mutate through either name and both see it
- a real copy: `a.copy()`

list

```
l = [123, 'spam', 1.23]
```

Operation	Description	Output
`len(l)`	length of the list	`3`
`l[1]`	indexing, 0-based	`'spam'`
`l[-1]`	backwards indexing	`1.23`
`l[0:2]`	slicing, extracts a sublist	`[123, 'spam']`
`l + [4, 5, 6]`	concatenation	`[123, 'spam', 1.23, 4, 5, 6]`
`l * 2`	repetition	`[123, 'spam', 1.23, 123, 'spam', 1.23]`

HW1

- released today on the course website and Canvas
- due next Thursday, Sep 10
- submit on Canvas as an .ipynb file
- general rule
  - you should only need to use content up until the day of release date

next time

- Thursday
  - control flow & dict
    - *Think Python*, Ch. 5, §§5.1–5.7: Conditionals
    - *Think Python*, Ch. 7, §§7.1 and 7.3–7.5: Loops, Updating, Counting
    - *Think Python*, Ch. 10, §§10.1–10.5: Dictionaries and Counters

logistics

- classes on week 6 (Sep 29 and Oct 1) will be asynchronous