Python Tutorial 2

LING-381-Language Technology and LLMs

Instructor: Hakyung Sung

September 11, 2025

Review: Tutorial 1

How to learn to ride a bike is ...

Review: Tutorial 1

to actually ride a bike.

Review: Basic value types

Python has several basic types of values. The most common include:

- **Strings** (text)

Review: Basic value types

Python has several basic types of values. The most common include:

- **Strings** (text)
- **Integers** (whole numbers)

Review: Basic value types

Python has several basic types of values. The most common include:

- **Strings** (text)
- **Integers** (whole numbers)
- **Floats** (decimal numbers)

Review: Basic value types

Python has several basic types of values. The most common include:

- **Strings** (text)
- **Integers** (whole numbers)
- **Floats** (decimal numbers)
- **Booleans** (`True` and `False`)

Review: Basic value types

Python has several basic types of values. The most common include:

- **Strings** (text)
- **Integers** (whole numbers)
- **Floats** (decimal numbers)
- **Booleans** (`True` and `False`)
- **None** (represents "no value")

Review: Store values

Values

- Store values of any type.

```
a = "thing is a string"
b = 9
c = 3.2
```

```
print (b+c)
```

```
12.2
```

```
print (a+"!")
```

```
thing is a string!
```

Review: Functions

Functions

```python
print("Hi!")
```

```
Hi!
```

```python
len("abc")
```

```
3
```

```python
str(16)
```

```
'16'
```

```python
int(3.9)
```

```
3
```

```python
float("2.5")
```

```
2.5
```

```python
type("abc") # you can check the type of the variables using this function
```

```
str
```

```python
type(16)
```

```
int
```

Review: Methods

Methods

```
sample = "This is a STRING"
```

```
sample.lower()
```

```
'this is a string'
```

```
sample.lower().split(" ")
```

```
['this', 'is', 'a', 'string']
```

```
sample = "This, is, a, STRING"
```

```
# How can we modify this line?
```

```
['this', 'is', 'a', 'string']
```

Review: Membership test

Membership test

- Use the `in` operator to check if a substring exists:

```python
word = "awesome"

if "a" in word:
  print ("Contains 'a'!")
else:
  print("No 'a' found.")
```

```
Contains 'a'!
```

Tip: use `elif` for checing additiona conditions

- elif = "else if"
- Used to check additional conditions after an initial if statement.
- only checked if the first condition was False

```python
word = "awesome"

# we now want to check 'e' as well.
if "a" in word:
  print ("Contains 'a'")
elif "e" in word:
  print ("Contains 'e'")
else:
  print("No 'a' or 'e' found.")
```

```
Contains 'a'
```

Review: for and while loop

Loops

```
words = ["read", "code", "repeat"]

for w in words:
    print(w)
```

```
read
code
repeat
```

```
for w in words:
  if "e" in w:
    print(w)
```

```
read
code
repeat
```

```
count = 0
while count < 3:
    print(count)
    count += 1
```

```
0
1
2
```

```
while w in words:
  print(w)
```

Review: Tuple

3. tuples, dictionaries, functions, classes, save into files

```python
# Create and concatenate
t = (1, 2)
t = t + (3, 4)
print(t)
```

```
(1, 2, 3, 4)
```

```python
# Indexing and slicing
print(t[0])
print(t[-1])
print(t[1:3])
```

```
1
4
(2, 3)
```

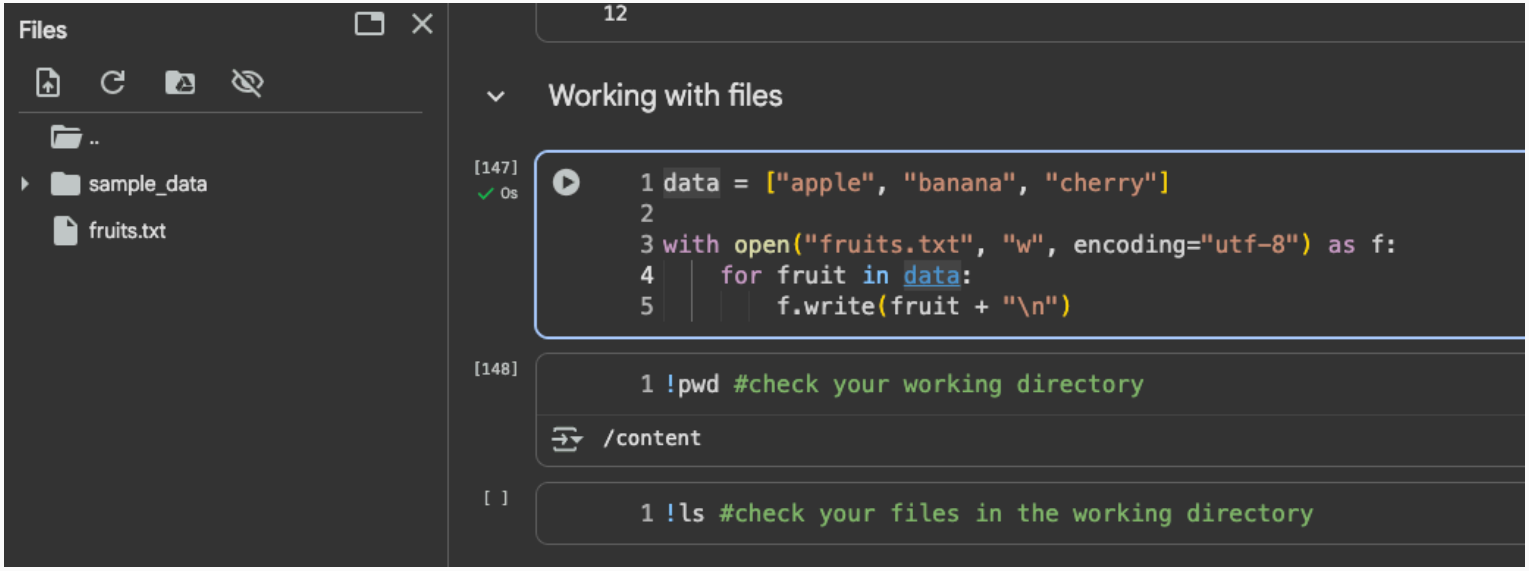
```python
# Unpacking
x, y, *rest = t
print(x, y, rest)
# Explanation:
# x gets the first value (1), y gets the second value (2),
# and *rest collects all the remaining values into a list → [3, 4].
```

```
1 2 [3, 4]
```

```python
*begin, last = t
print(begin, last)

first, *middle, last = t
print(first, middle, last)
```

Review: Working with files



Tutorial 2

- We often collect text data from the web.

Tutorial 2

- We often collect text data from the web.
- Raw text can be **messy**: strange symbols, broken characters, odd formatting.

Tutorial 2

- We often collect text data from the web.
- Raw text can be **messy**: strange symbols, broken characters, odd formatting.
- If we put messy text into a *`DataFrame`*, it may look confusing or inconsistent.

Tutorial 2

- We often collect text data from the web.
- Raw text can be **messy**: strange symbols, broken characters, odd formatting.
- If we put messy text into a *`DataFrame`*, it may look confusing or inconsistent.
- **First step: clean the text** so that we can work with it safely.

Tutorial 2

- We often collect text data from the web.
- Raw text can be **messy**: strange symbols, broken characters, odd formatting.
- If we put messy text into a *`DataFrame`*, it may look confusing or inconsistent.
- **First step: clean the text** so that we can work with it safely.
- After cleaning, we can move on to:
  - **Tokenization**

Tutorial 2

- We often collect text data from the web.
- Raw text can be **messy**: strange symbols, broken characters, odd formatting.
- If we put messy text into a *`DataFrame`*, it may look confusing or inconsistent.
- **First step: clean the text** so that we can work with it safely.
- After cleaning, we can move on to:
  - **Tokenization**
  - **Lemmatization**

Preview

- **Tokenization**: Converting raw text into tokens (words, punctuation, numbers). Essential for breaking text into analyzable units.

Preview

- **Tokenization**: Converting raw text into tokens (words, punctuation, numbers). Essential for breaking text into analyzable units.
- **Lemmatization**: Converting each token to its base (dictionary) form.

Preview

- **Tokenization**: Converting raw text into tokens (words, punctuation, numbers). Essential for breaking text into analyzable units.
- **Lemmatization**: Converting each token to its base (dictionary) form.
- **Frequency Calculation**: Counting token/lemma occurrences in a text.

Preview

- **Tokenization**: Converting raw text into tokens (words, punctuation, numbers). Essential for breaking text into analyzable units.
- **Lemmatization**: Converting each token to its base (dictionary) form.
- **Frequency Calculation**: Counting token/lemma occurrences in a text.
- **Concordance**: Displaying each occurrence of a word/phrase with its surrounding context.

Preview

- **Tokenization**: Converting raw text into tokens (words, punctuation, numbers). Essential for breaking text into analyzable units.
- **Lemmatization**: Converting each token to its base (dictionary) form.
- **Frequency Calculation**: Counting token/lemma occurrences in a text.
- **Concordance**: Displaying each occurrence of a word/phrase with its surrounding context.

Now, let's get our hands dirty!