LG 467 Computers in Linguistics

[1-2021] Python 2: Python Basics

Sakol Suethanapornkul

Recap

Previously...

- Anaconda Navigator
- Spyder (iPython Interpreter and Editor)
  - Interpreter: try codes and see results immediately
  - Editor: create and edit codes

Previously...

Python as a calculator (i.e., arithmetic)

```
5 + 3
7 - 2

4 / 2
7 // 2
7 % 2
4 ** 2

5 + 3 * 8
4 - 2 / 3

(5 + 3) * 8
5 + (3 * 8)
4 - (2 / 3)
```

Code 0.1

Previously...

We created a class folder and learned how to save a file (.py)

We saw how to create `variables` and how to call `functions`

```
greet = "Hello, world!"
```

Code 0.2

```
print(greet)
```

Code 0.3

It’s time to dive deeper!

Overview

What does a computer do?

Fundamentally, a computer

- performs (trillions of) operations
- remembers results

But computers only know what you tell them!

- You can tell computers what to do with a program
- Programs are like a recipe (a set of "how-to" instructions)

Adapted from: MIT's 6.0001 online lecture

What is needed in a recipe?

Ingredients (for 12 cookies)

½ cup granulated sugar (100 g)

¾ cup brown sugar (165 g), packed

1 teaspoon salt

½ cup unsalted butter (115 g), melted

1 egg

1 teaspoon vanilla extract

1¼ cups all-purpose flour (155 g)....

Preparation

1. Whisk together the sugars, salt, and butter
2. Whisk in the egg and vanilla
3. Sift in the flour and baking soda
4. Then fold the mixture with a spatula
5. Fold in the chocolate chunks
6. chill the dough for at least 30 minutes
7. Scoop the dough onto a baking sheet

Adapted from: Tasty's chocolate cookie recipe

What is needed in a recipe?

- A **sequence of** simple **steps**
- **Flow of control** process that specifies when each step is executed
- A means of determining **when to stop**

1 + 2 + 3 = an algorithm

Adapted from: MIT's 6.0001 online lecture

A recipe

Code 1.1

```
txt = '''Beautiful is better than ugly. Explicit is better than
implicit. Simple is better than complex. Complex is better than
complicated. Flat is better than nested.'''

txt_ls = txt.split()

l = []

for word in txt_ls:
    if len(word) > 7:
        l.append(word)

#or equivalently
m = [word for word in txt_ls if len(word) > 7]

print(l)
print(m)
```

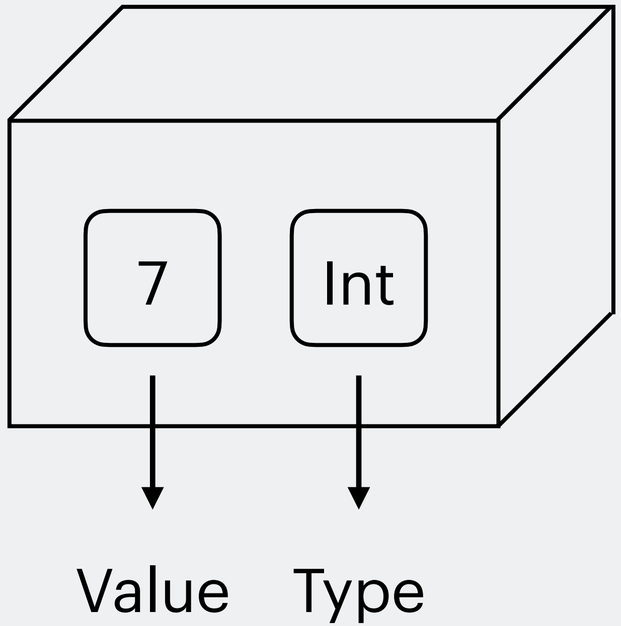
Python Basics

Data: types, values, variables, names

Python data are objects

- Python bundles a value with a type of that value → an object

Analogy:



Data: types, values, variables, names

Interpreter: `type()`

Code 1.2

```
type(-7)
type(7 * 2)
type(10e2)

type(3.1415)

type("cool")

type(True)
```

Question: How many types are there?

Data: types, values, variables, names

Values tell you something about types

5; 32; 45000	→	integers	`type(5)`	`int`
18.25; 3.1415	→	floating points	`type(3.1415)`	`float`
True; False	→	Booleans	`type(True)`	`bool`
"cat"; "girls"	→	strings	`type("cat")`	`str`

Data: types, values, variables, names

Type conversion: `str()`, `int()`, `float()`, `bool()`

Code 1.3

```
str(7)
int(3.83)
float(12)

str(True)
int(True)

# Will this work?
int("cat")
```

NOTE: For booleans, True = 1; False = 0

Data: types, values, variables, names

We'd like to "save" data objects for later use. Let’s create `variables`

- `Variables` = **names** for data objects

Code 1.4

```
num = 5
an1 = 'cat'
an2 = "dog"
an3 = '''horse'''

pi_approx = 3.14159
```

Code 1.5

```
# what's going to happen here?
num =
```

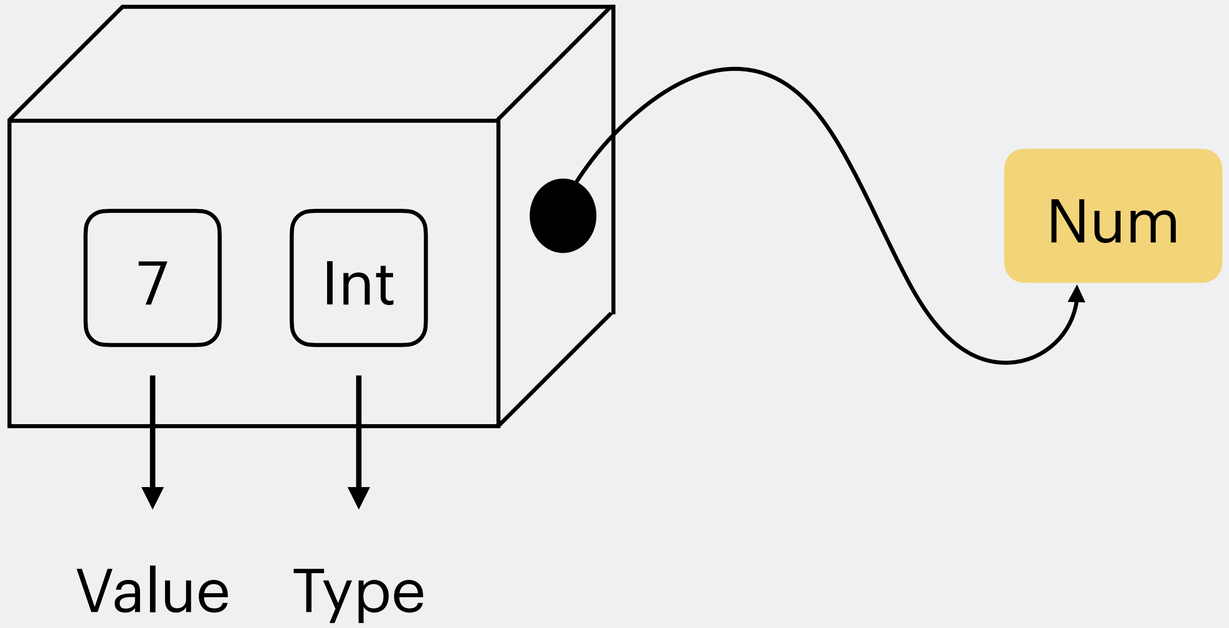
Data: types, values, variables, names

Assignment attaches a **name** to an object

```
num = 7
```

Code 1.6

Analogy:



Data: types, values, variables, names

Question: Which of the following are variable names in Python?

Code 1.7

```
name     = 'sakol'
nms_inst = 'sakol'
nms.inst = 'sakol'
names1   = 'sakol'
1names   = 'sakol'
NameInstr = 'sakol'
names@1   = 'sakol'
Name_Inst = 'sakol'
```

Code 1.8

```
# variable names can’t be reserved words. check:
help("keywords")
```

Data: types, values, variables, names

PEP-8: style guide for Python code (the Pythonic way)

1. Variables and functions should have informative, lower case names:
   - ✔ `word_count` ✘ `wrdct`
   - ✔ `find_verbs()` ✘ `Find_Verbs()` or `Fnd_vrb()`

2. White space around operators
   - ✔ `word_count = word_count + 1` ✘ `word_count=word_count+1`

3. Line length should be 79 characters maximum

Data: types, values, variables, names

Copying: assigning two names to the same object

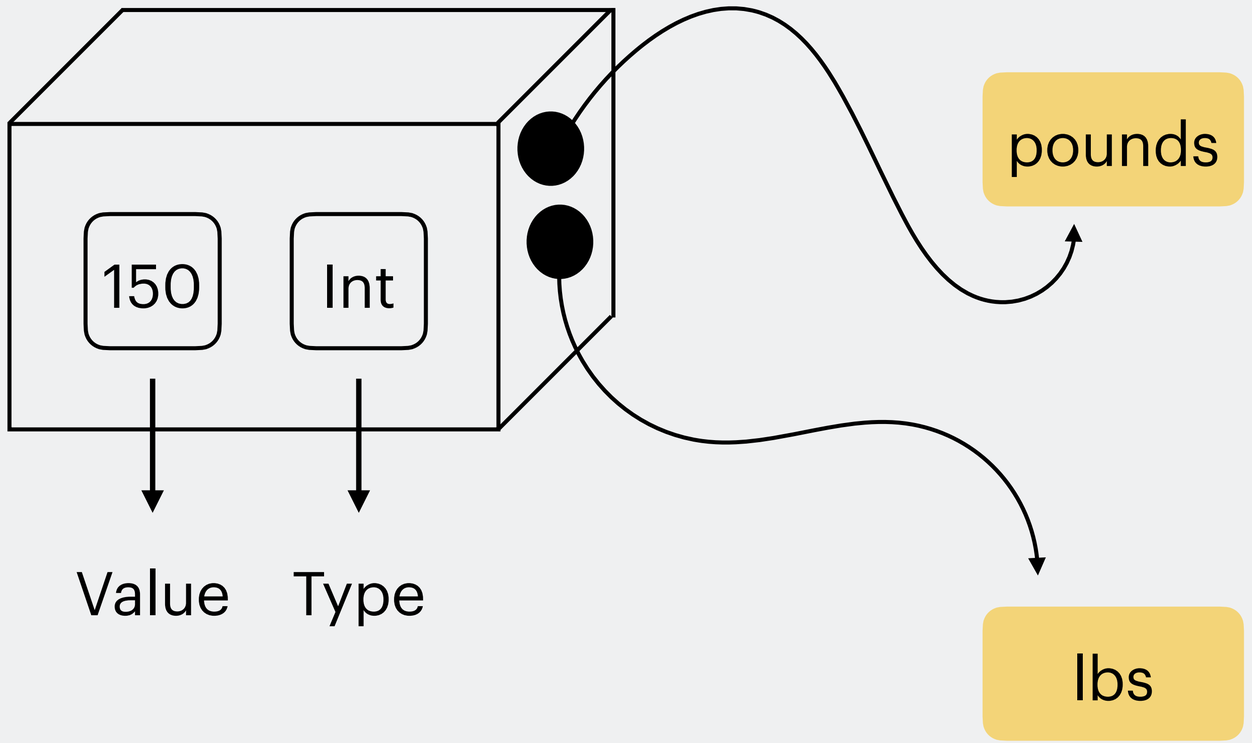
Code 1.9

```
pounds = 150
lbs    = pounds

pounds = 175

pounds # or print(pounds)
lbs    # which value?
```

Data: types, values, variables, names

Copying: assigning two names to the same object

Analogy:



Data: types, values, variables, names

But beware. `Lists` are mutable.

Code 1.10

```
names    = ['prayut', 'anutin', 'somsak', 'tummanat']
cabinet  = names

names[0] = 'apisit'

names        # or print(names)
cabinet
```

We'll talk about lists in future classes!

String variables

As linguists, we work with texts. Python handles texts as strings.

- A string is a **sequence** of **characters**

Code 1.11

```
# Create string variables
name  = 'Sakol'

store = "Teddy's bigger burger"

poem  = '''Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth'''
```

String variables

Escape characters: Precede characters with \ to achieve effects

Code 1.12

```
# \n for newlines
print("cars, \nMars, \nand some vars")

# \t for tabs
print("cars, \tMars, \tand some vars")

# \" to have double quotes inside double quotes
print("He said: \"I don't know\"")
```

String concatenation

We can combine and duplicate string variables

Code 1.13

```
# Let's start with the basics
x = 10
x + 3
x ** 3

# How about strings
rs = "npr"
tv = "pbs"
rs + tv
rs + " " + tv
rs * 3
rs / 3  # What's going to happen here?
```

Aside: `print()`

The `print()` function is meant for human output

- `print()` adds spaces and newlines

Code 1.14

```
# print() a function with arguments inside parentheses
print(rs, tv)
print(rs,tv)

# This is similar to rs + tv
print(rs + tv)

# Build custom message inside print()
print("I get my news from", rs, "and", tv)
```

String indexing & slicing

To extract elements from a string variable, specify offsets inside [ ]

String:	i	P	h	o	n	e	_	1	2
Offset:	0	1	2	3	4	5	6	7	8
					... −5	−4	−3	−2	−1

Code 1.15

```
# Here's in code
phone = "iPhone 12"
phone[0]
phone[3]
phone[-1]
```

String indexing & slicing

To extract a substring, we can add more information inside [ ]

- [start: end: step]

Code 1.16

```
# Start is inclusive but end is exclusive (x-1)
phone = "iPhone 12"
phone[0:3]
phone[0: ]
phone[ :3]
phone[-4:-1]
phone[-4:  ]
phone[2:-2]
phone[0:7:2]
phone[::-1]
```

String methods

Useful built-in functions that work specifically with string variables

Code 1.17

```
# .split() returns a list
poem  = '''Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth'''

poem.split('\n')
poem.split(' ')   # equivalent to poem.split()
```

A more sophisticated way to split strings later on in the course

String methods

Useful built-in functions that work specifically with string variables

Code 1.18

```
fairy = "Once upon a time, in a far, far away land.
There's a charming princess."

# Case
fairy.lower()
fairy.upper()
fairy.capitalize()

# Replacing: ('old', 'new')
fairy.replace('time', 'century')

# Assign to the original variable
fairy = fairy.upper()
```

String methods

Useful built-in functions that work specifically with string variables

```
# String multiple methods together
fairy.lower().replace('time', 'century')

# But watch out replace() needs an exact match
fairy.upper()
fairy.upper().replace('time', 'century')
fairy.upper().replace('A', ‘AN’)

# This isn't efficient but does the job
fairy.replace('.', '!').replace(',', ';')
```

Code 1.18

[Continued]

String methods

Useful built-in functions that work specifically with string variables

```
# Frequency counts (exact match)
fairy.count('far')
fairy.count('Far') # Different from the above

# Remove leading & trailing spaces
ad = "     Privacy, simplified    "
ad.strip()

curse = "What the ****!!!?"
curse.strip('*!?')
```

Code 1.18

[Continued]

String methods

Lots of string methods we haven’t covered, but before we move on

```
# Because these methods work with strings...
'Once upon a time, in a far, far away
land'.count('far')

'''Two roads diverged in a wood, and I-
I took the one less traveled by,
And that has made all the
difference.'''.lower().split()

# And you can embed everything inside print()
print(fairy.lower().split())
print('Once upon a time'.lower().split())
```

Code 1.18

[Continued]

More functions

Plenty of built-in functions: `len()`, `print()`, `input()`

Code 1.19

```
len(fairy)

# input() get users to input data into Python
age = input("Enter your age: “)

print("How old are you? Enter your age")
x = input()

# input() and print() are basic I/O in Python
age = input("Enter your age: “)
print("You're", age, "years old!")
```

For more information

For more string methods:

- Chapter 5 in Lubanovich (2020) *Introducing Python*
- W3 Schools' website

Our plan next week!

Language and Computer, Chapter 3

- Sections 3.3 and 3.4 (Tokenization: What is it? What is it for?)