Chapter 14
Classes and Functions

A. Downey, *Think Python: How to Think Like a Computer Science,* 3rd ed., O'Reilly, 2024.

https://allendowney.github.io/ThinkPython/

Contents

1. Programmer-Defined Types

- We have used many of Python's built-in types—now we will define a new type. A programmer-defined type is also called a **class**.
- We'll create a type called **Time** that represents a time of day.

```
class Time:
    """Represents a time of day."""
```

  - The header indicates that the new class is called **Time**. The body is a docstring that explains what the class is for. Defining a class creates a **class object**.

- The class object is like a factory for creating **objects**. To create a **Time** object, you call **Time** as if it were a function:

```
lunch = Time()
type(lunch)
```

```
__main__.Time
```

```
print(lunch)
```

```
<__main__.Time object at 0x7f79449aa330>
```

1. Programmer-Defined Types

- Creating a new object is called **instantiation**, and the object is an **instance** of the class.

2. Attributes

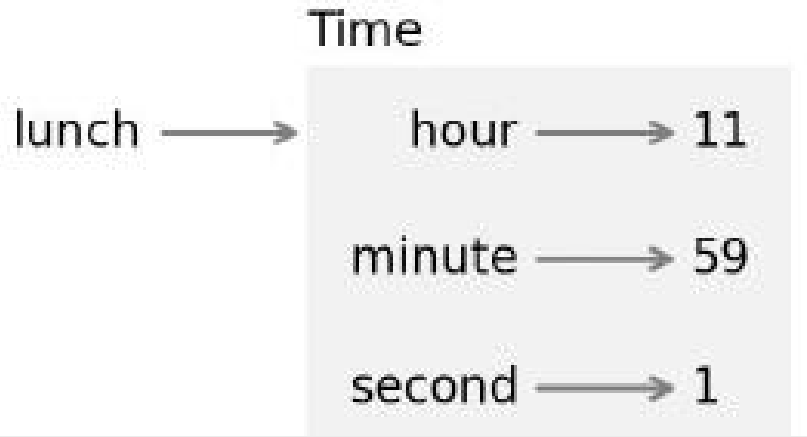
- An object can contain variables, which are called **attributes**. We can create attributes using **dot notation**:

```
class Time:
    """Represents a time of day."""
```

```
lunch = Time()
```

```
lunch.hour = 11
lunch.minute = 59
lunch.second = 1
```

- Object diagram



2. Attributes

- You can read the value of an attribute using the dot operator:
  - You can use an attribute as part of any expression or in an expression in an f-string.

```
class Time:
    """Represents a time of day."""
```

```
lunch = Time()
```

```
lunch.hour = 11
lunch.minute = 59
lunch.second = 1
```

```
f'{lunch.hour}:{lunch.minute}:{lunch.second}'
```

```
'11:59:1'
```

```
f'{lunch.hour}:{lunch.minute:02d}:{lunch.second:02d}'
```

```
'11:59:01'
```

3. Objects as Return Values

- Functions can return objects.

```
class Time:
    """Represents a time of day."""
```

```
def print_time(time):
    s = f'{time.hour:02d}:{time.minute:02d}:{time.second:02d}'
    print(s)

def make_time(hour, minute, second):
    time = Time()
    time.hour = hour
    time.minute = minute
    time.second = second
    return time
```

```
time = make_time(11, 59, 1)
print_time(time)
```

```
11:59:01
```

4. Objects Are Mutable

```
class Time:
    """Represents a time of day."""
```

```
def print_time(time):
    s = f'{time.hour:02d}:{time.minute:02d}:{time.second:02d}'
    print(s)

def make_time(hour, minute, second):
    time = Time()
    time.hour = hour
    time.minute = minute
    time.second = second
    return time
```

```
start = make_time(9, 20, 0)
print_time(start)
```

```
09:20:00
```

```
start.hour += 1
start.minute += 32
print_time(start)
```

```
10:52:00
```

4. Objects Are Mutable

```
class Time:
    """Represents a time of day."""
```

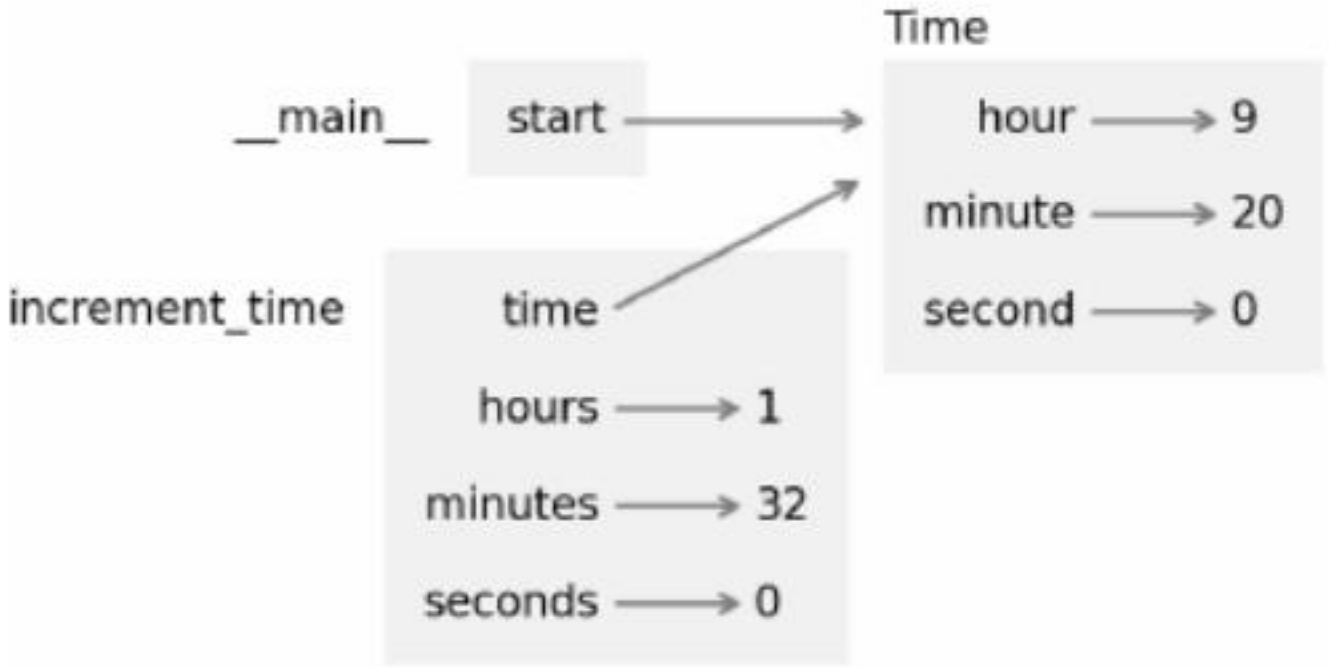
```
def print_time(time):
    s = f'{time.hour:02d}:{time.minute:02d}:{time.second:02d}'
    print(s)

def make_time(hour, minute, second):
    time = Time()
    time.hour = hour
    time.minute = minute
    time.second = second
    return time

def increment_time(time, hours, minutes, seconds):
    time.hour += hours
    time.minute += minutes
    time.second += seconds
```

```
start = make_time(9, 20, 0)
increment_time(start, 1, 32, 0)
print_time(start)
```

```
10:52:00
```

4. Objects Are Mutable

- The following stack diagram shows the state of the program just before **increment_time** modifies the object:



5. Copying

- The **copy** module provides a function called **copy** that can duplicate any object.
  - Create a new object which contains the same data. (they are not the same object)

```
from copy import copy

start = make_time(9, 20, 0)
end = copy(start)

print_time(start)
print_time(end)
```

```
09:20:00
09:20:00
```

```
start is end
```

```
False
```

5. Copying

- Let's see what the **==** operator does.
  - You might expect **==** to yield **True** because the objects contain the same data.
  - But for programmer-defined classes, the default behavior of the **==** operator is the same as the **is** operator—it checks identity, not equivalence.

```
from copy import copy

start = make_time(9, 20, 0)
end = copy(start)

print_time(start)
print_time(end)
```

```
09:20:00
09:20:00
```

```
start is end
```

```
False
```

```
start == end
```

```
False
```

6. Pure Functions

- We can use **copy** to write pure functions that don't modify their parameters.

```
def add_time(time, hours, minutes, seconds):
    total = copy(time)
    increment_time(total, hours, minutes, seconds)
    return total
```

```
start = make_time(9, 20, 0)
end = add_time(start, 1, 32, 0)
print_time(end)
```

```
10:52:00
```

  - The return value is a new object representing the end time. And we can confirm that start is unchanged.

```
print_time(start)
```

```
09:20:00
```

6. Pure Functions

- **add_time** is a pure function because it does not modify any of the objects passed to it as arguments and its only effect is to return a value.
- Anything that can be done with modifiers can also be done with pure functions. In fact, some programming languages only allow pure functions.
- Programs that use pure functions might be less error prone than programs that use modifiers. But modifiers are sometimes convenient and can be more efficient.
- In general, I suggest you write pure functions whenever it is reasonable and resort to modifiers only if there is a compelling advantage. This approach might be called a **functional programming style**.