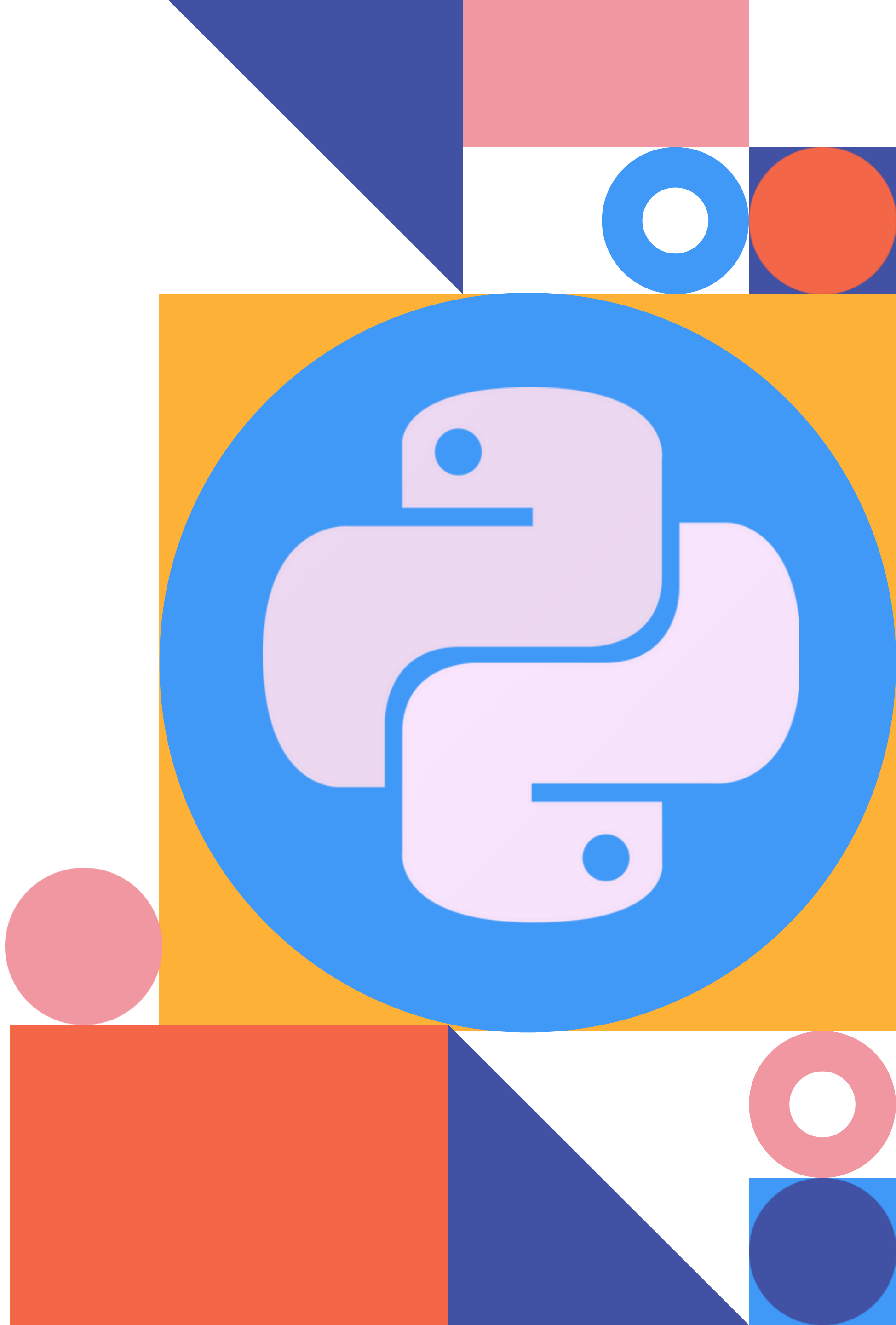
Object-Oriented Programming

Yan-Fu Kuo

Dept. of Biomechatronics Engineering
National Taiwan University

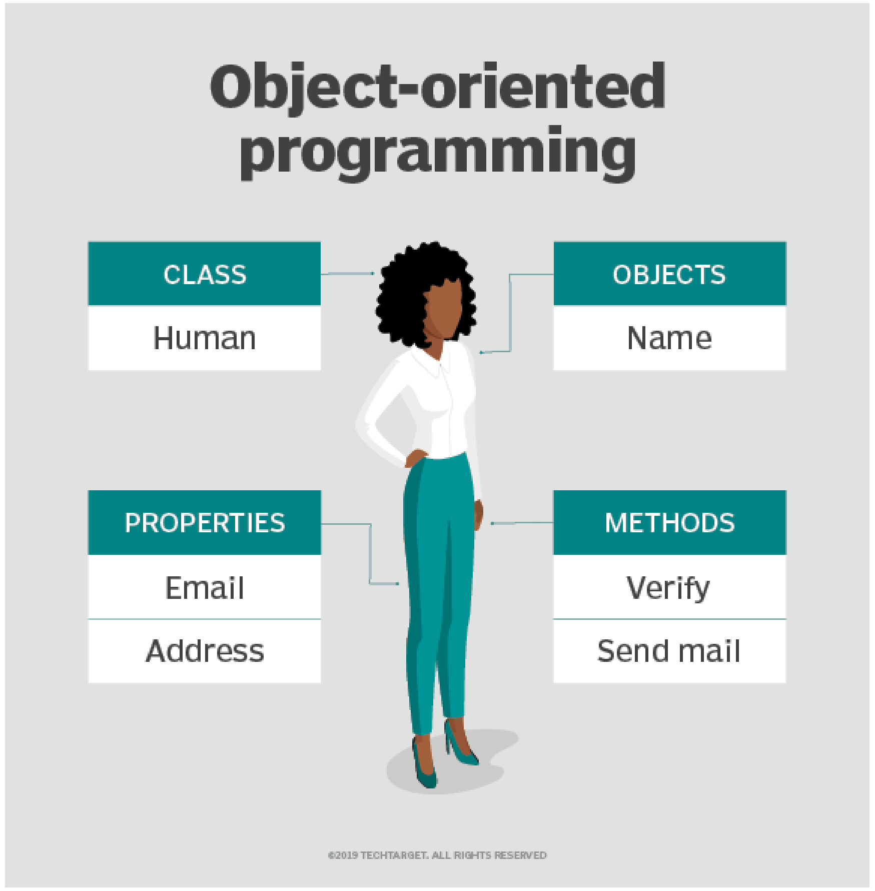
→



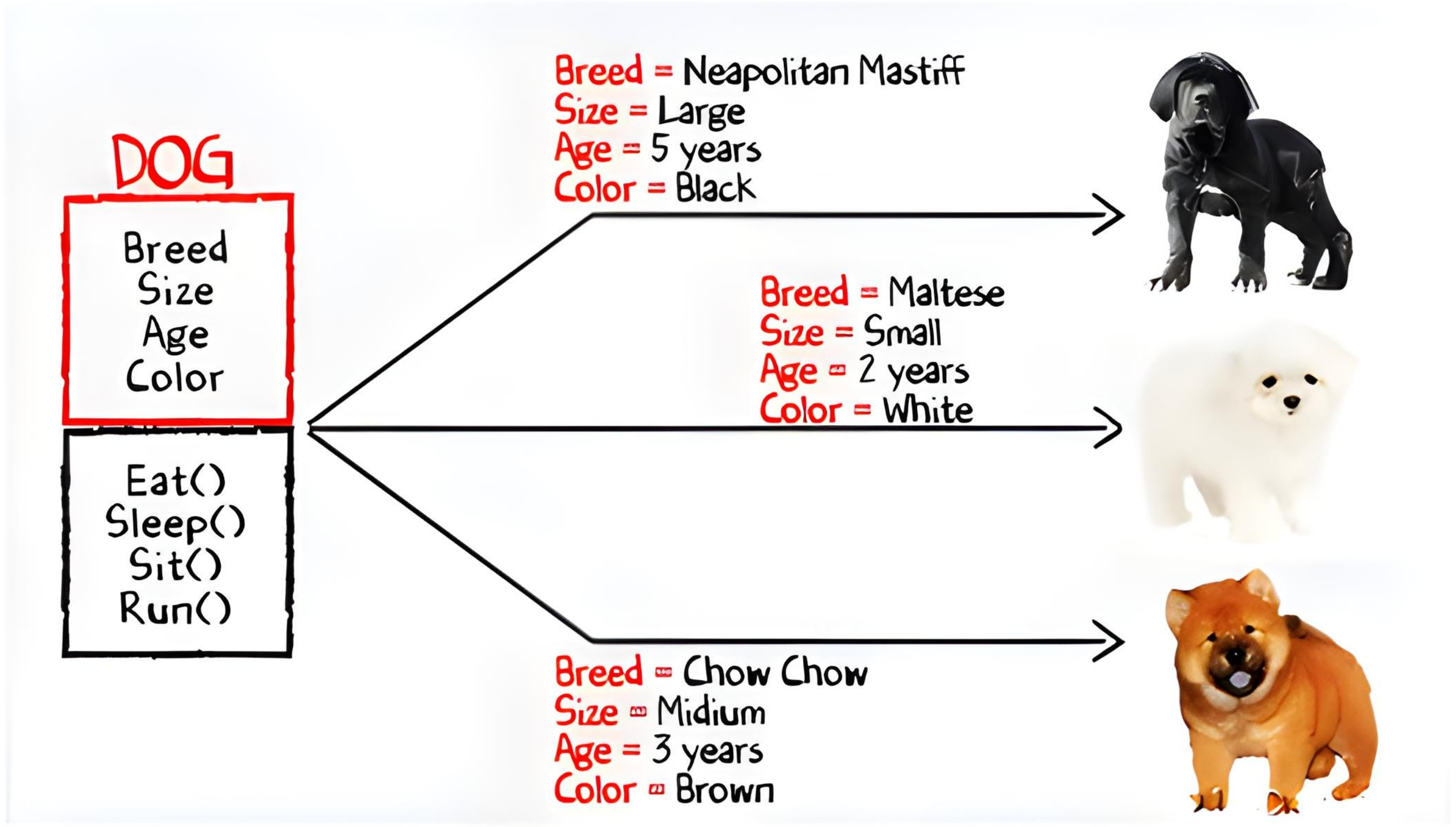
Contents

What Is An Object?

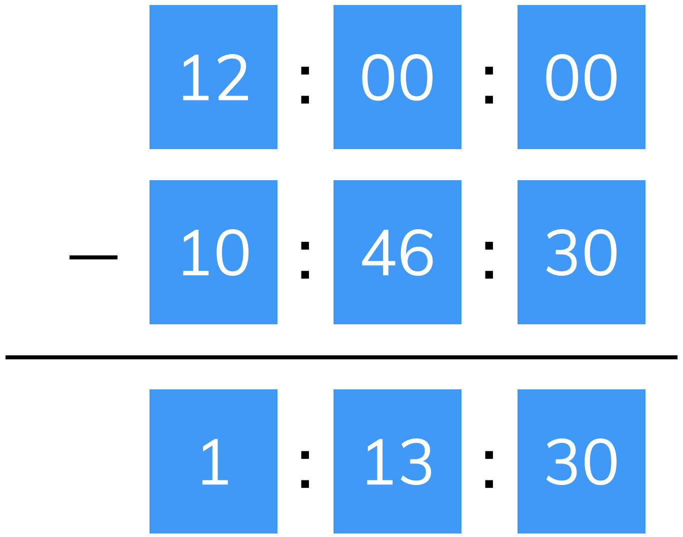
Everything in python is an object – classes, functions, and even simple data types, such as integer and float.



Object vs. Class



Scenario – An Object Storing `Time`



Variable `hour = 12`
Variable `minute = 00`
Variable `second = 00`

Can be stored in tuple
`now = (12, 00, 00)`

Create this new type as an object

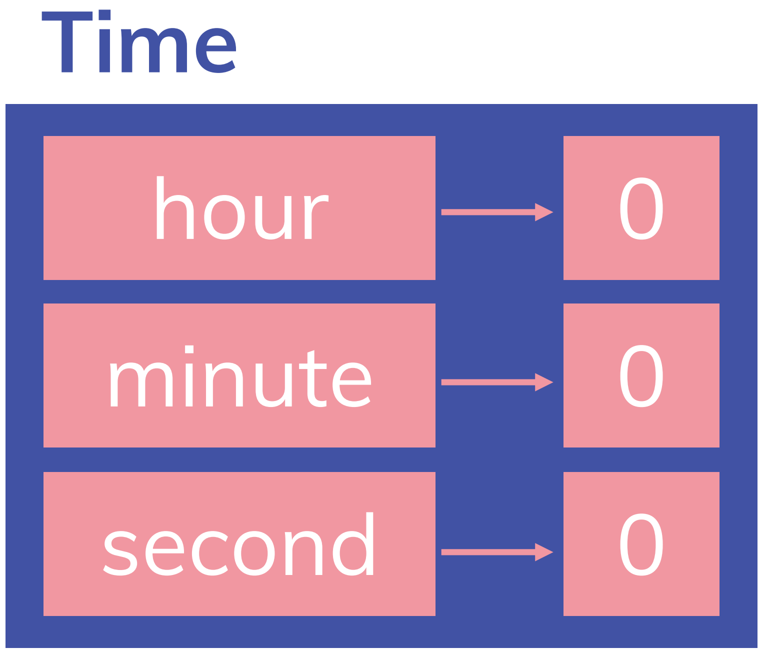
Example of User-defined Object: `Time`

Keyword for user-defined objects: **class**

```
class Time:
  hour = 0
  minute = 0
  second = 0
```

```
now = Time()
now
```

```
<__main__.Time object at 0x00CA848>
```



Data Type of User-defined Object

Predefined

```
x = 3
type(x)
```

```
<class 'int'>
```

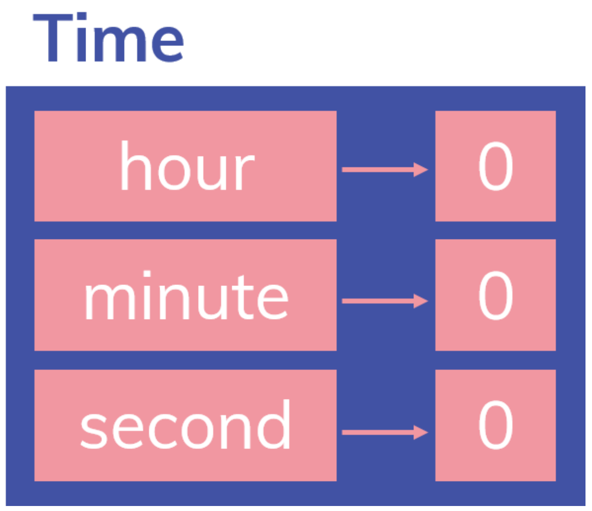
User-defined

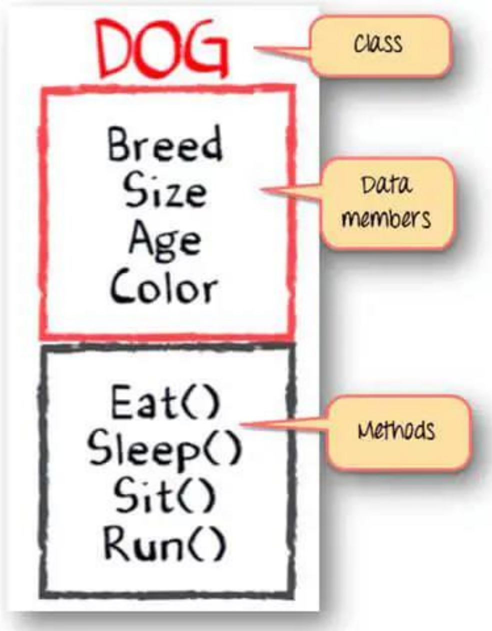
```
now = Time()
type(now)
```

```
<class 'Time'>
```

Class Members – Attributes and Methods

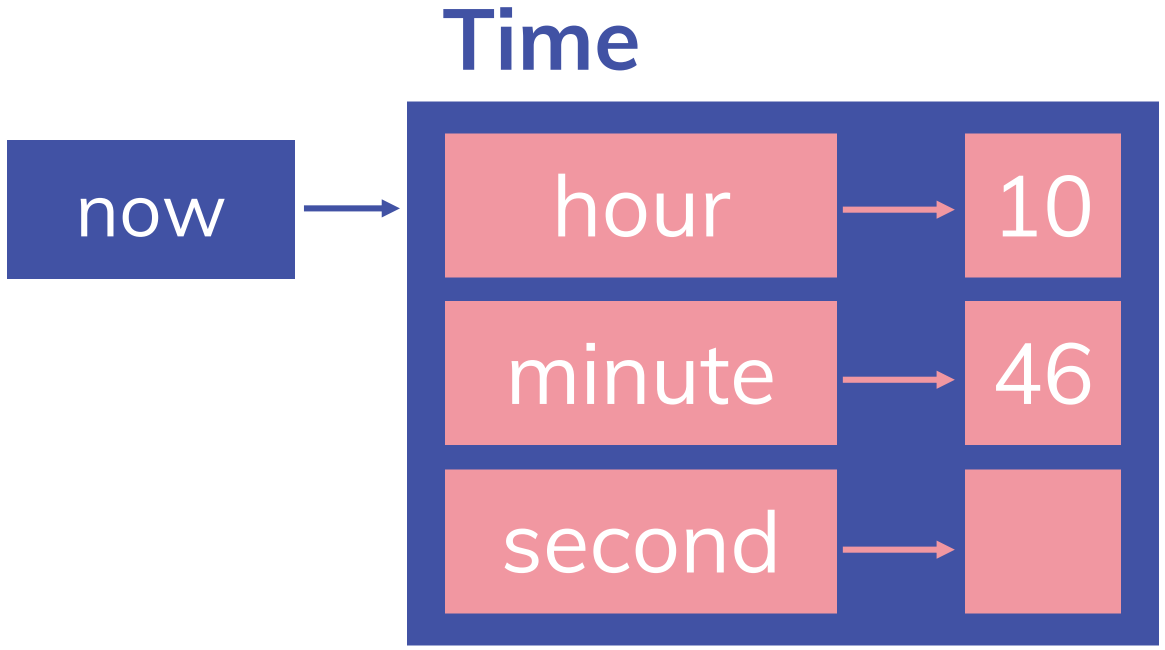
- Attributes are variables built in an object.
- May not be accessible from outside.





Accessing Class Attributes

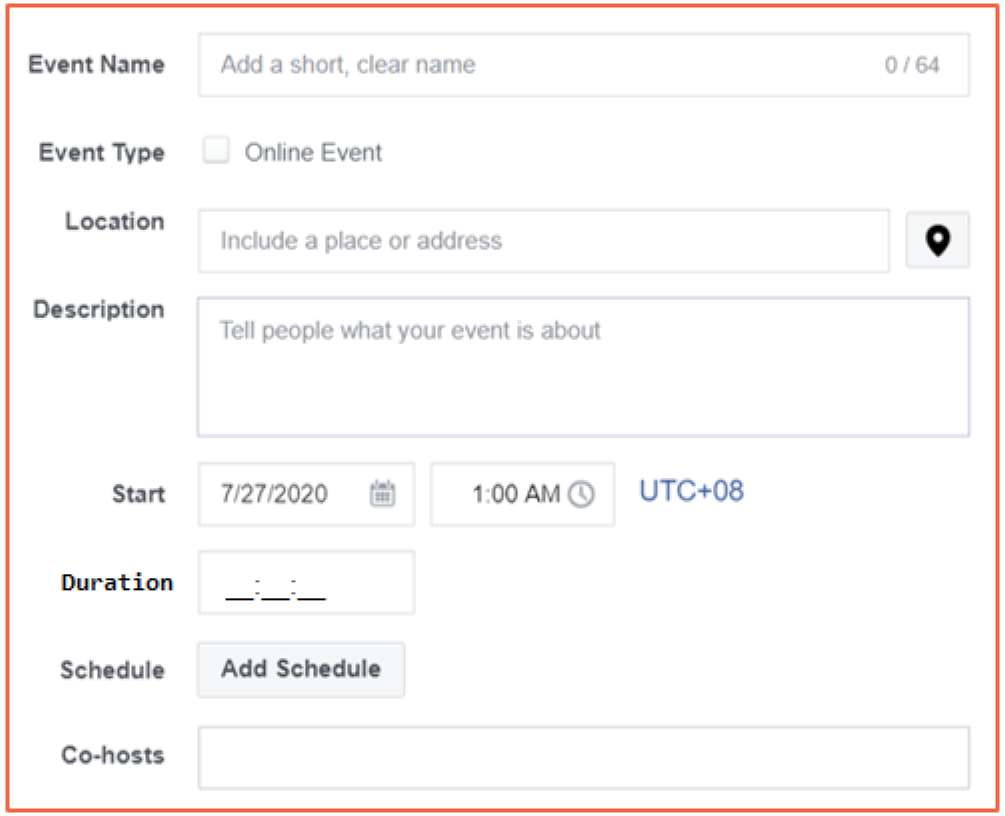
```
>>> now = Time()
>>> now.hour = 10
>>> now.minute = 46

>>> now.hour
10
>>> now.minute
46
```



Class Inherit – Using **Event** as an Example

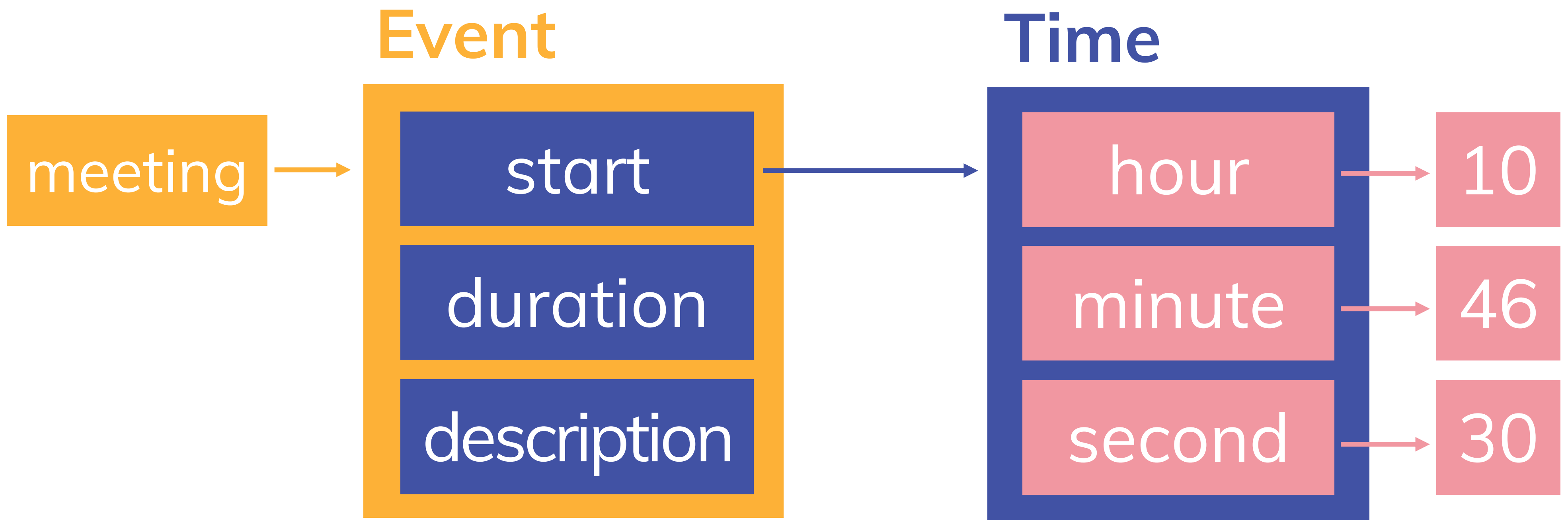
	Name	Datatype
	Event name	`str`
	Event Type	`boolean`
	Location	`str`
✓	Description	`str`
✓	Start	**`Time`**
✓	Duration	`int`
	Co-hosts	`list`

Event



Event Name	Add a short, clear name 0 / 64
Event Type	☐ Online Event
Location	Include a place or address
Description	Tell people what your event is about
Start	7/27/2020 1:00 AM UTC+08
Duration	__:__:__
Schedule	Add Schedule
Co-hosts	

Meeting → Event → Time

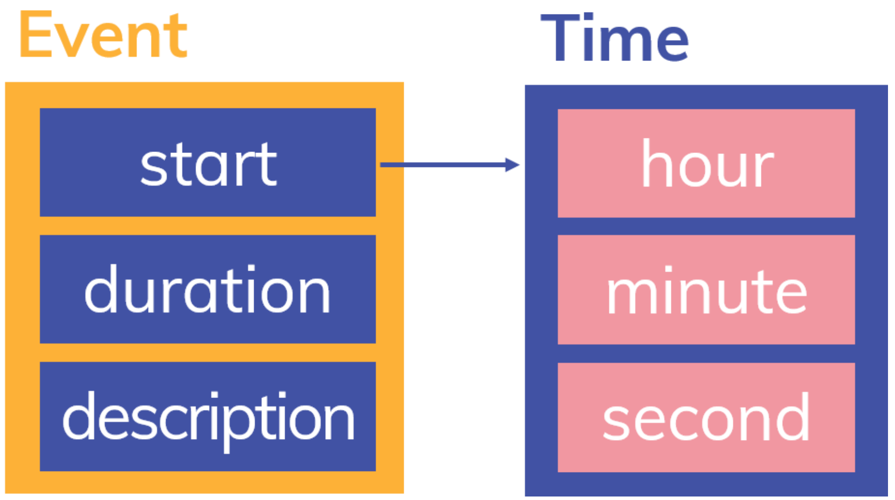


Assigning Attributes of Inherited Objects

Create a class Event.

```
class Event:
    ...
    ...
```

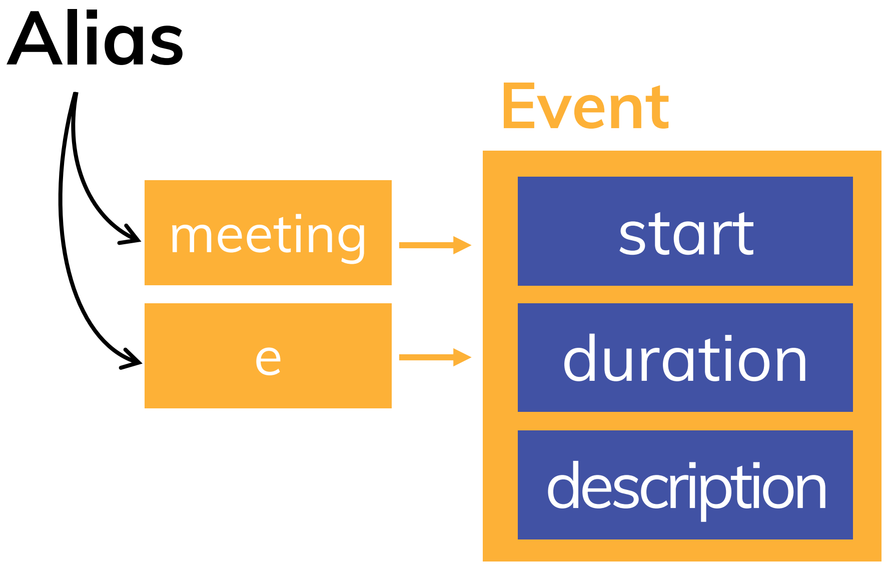
```
meeting = Event()
meeting.start = now
meeting.start.hour = 10
meeting.description = ''
```



Objects Are “Call by Reference”

```
def postpone_event(e, t):
  e.start.hour += t.hour
  e.start.minute += t.minute
  e.start.second += t.second
```

```
postpone_event(meeting, one_hour)
```



Aliasing can make a program difficult to read because changes in one place might have unexpected effects in another place.

Copy Objects

A built-in module to copy objects

```
>>> t1 = Time()
>>> t1.hour = 10
>>> t2 = t1
>>> t1 is t2
True
```

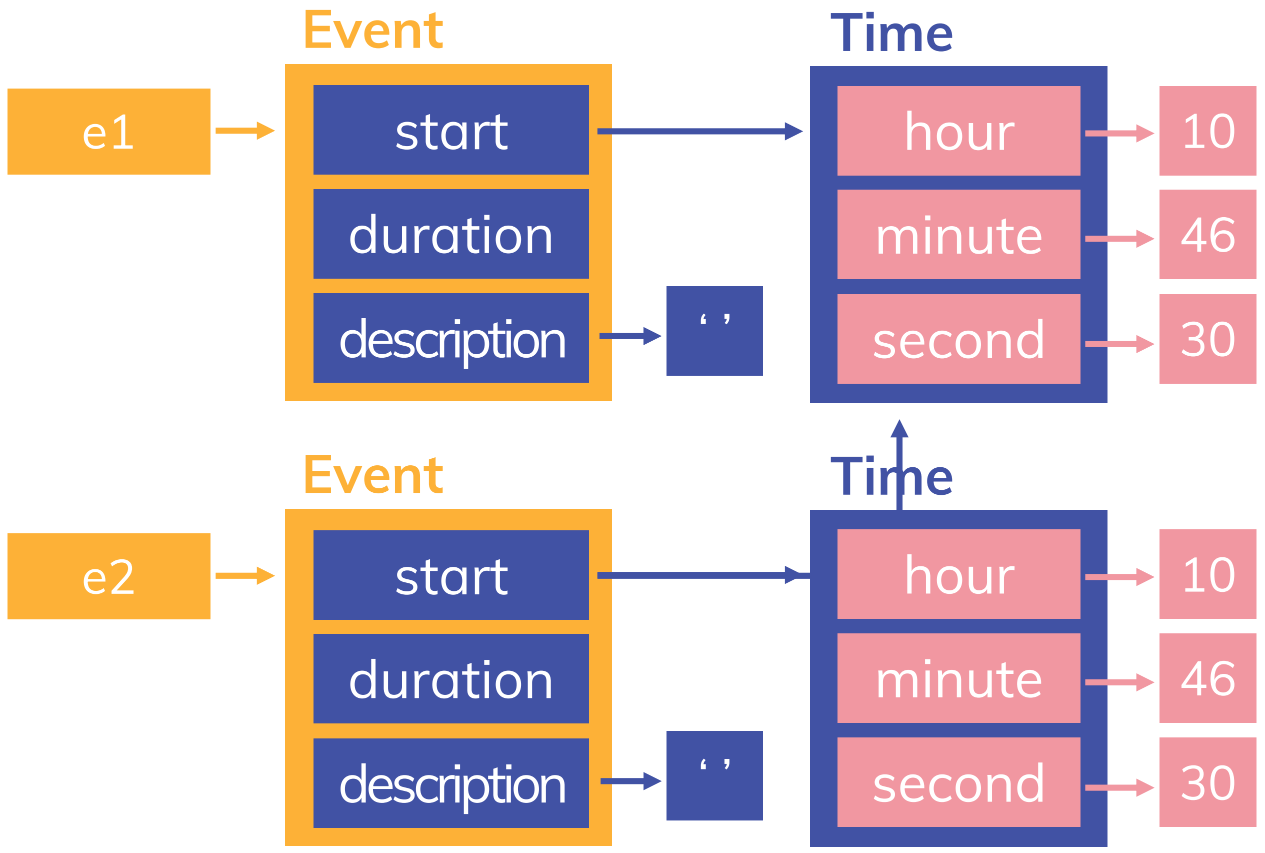
```
>>> import copy
>>> t2 = copy.copy(t1)
>>> t1 is t2
False
>>> t1 == t2
```

Shallow Copy vs. Deep Copy

```
>>> e1 = Event()
>>> e1.start = Time()
>>> e1.description =''
>>> e2 = copy.copy(e1)
>>> e1 is e2
False
```

```
>>> e1.start is e2.start
True
```

```
>>> e2 = copy.deepcopy(e1)
```



Debugging Tips

```
>>> meeting.location
AttributeError: 'Event' object has no attribute 'location'
```

```
>>> type(e1)
<class '__main__.Event'>
>>> isinstance(e1, Event)
True
>>> hasattr(e1, 'start')
True
>>> hasattr(e1, 'location')
False
```

```
try:
    x = meeting.location
except AttributeError:
    x = 0
```

Built-in functions (https://docs.python.org/3/library/functions.html)

Built-in Exceptions

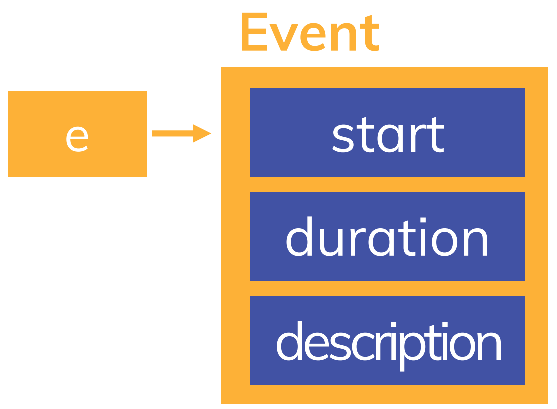
ArithmeticError	Raised when an error occurs in numeric calculations
AssertionError	Raised when an assert statement fails
AttributeError	Raised when attribute reference or assignment fails
Exception	Base class for all exceptions
EOFError	Raised when the input() method hits an "end of file" condition (EOF)
FloatingPointError	Raised when a floating point calculation fails
GeneratorExit	Raised when a generator is closed (with the close() method)
ImportError	Raised when an imported module does not exist
IndentationError	Raised when indendation is not correct
IndexError	Raised when an index of a sequence does not exist
KeyError	Raised when a key does not exist in a dictionary

https://www.w3schools.com/python/python_ref_exceptions.asp

A Pure Function Example – Adding Time

```
def add_time(t1, t2):
   sum = Time()
   sum.hour = t1.hour + t2.hour
   sum.minute = t1.minute + t2.minute
   sum.second = t1.second + t2.second
   return sum
```

Exercise



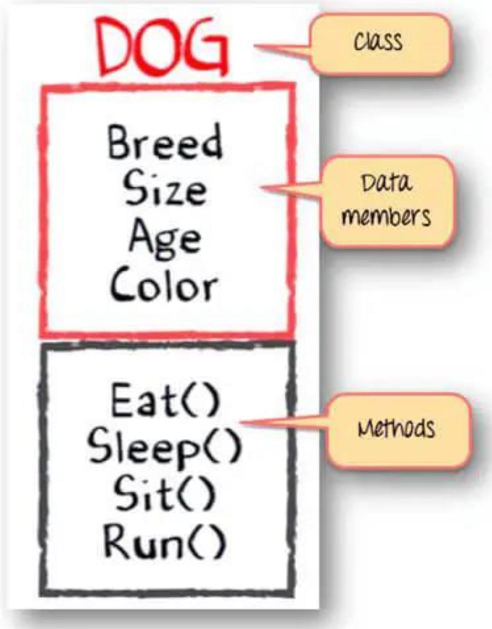
Write a function called get_end_time that takes an Event and returns the ending time of it.

```
def get_end_time(e):
  t = Time()
  ...
  return t
```

Class Members – Attributes and Methods

- Methods are functions built in an object.
- Mostly expressed in terms of operations on objects

```
def add_time(t1, t2):
  sum = Time()
  sum.hour = t1.hour + t2.hour
  sum.minute = t1.minute + t2.minute
  sum.second = t1.second + t2.second
  return sum
```



Class Methods

Methods are semantically the same as functions, but …

```
class Time():

def print_time(t1):

...
print_time(t1)
...
```

→

```
class Time():
    def print_time(t1):

...
print_time(t1)
...
```

A Function for Printing Time

```
>>> f'{now.hour}:{now.minute}:{now.second}'
'10:46:30'
```

```
def print_time(t):
  print(f'{t.hour}:{t.minute}:{t.second}')
```

```
>>> print_time(now)
'10:46:30'
```

f-string (https://docs.python.org/3/reference/lexical_analysis.html#f-strings)

self

- self represents the instance of the class.
- Accessing the attributes and methods of an object using self.

```
class check:
  def __init__(self):
    print("Address of self = ", id(self))
obj = check()
print("Address of class object = ", id(obj))
```

```
Address of self =  2885977398984
Address of class object =  2885977398984
```

Functions to Methods

Function

```
class Time:
    ...
def print_time(t):
    print(f'{t.hour}:{t.minute}:{t.second}')
```

```
>>> print_time(start)
```

Method (Wrong)

```
class Time:
    def print_time(t):
        print(f'{t.hour}:{t.minute}:{t.second}')
```

```
>>> Time.print_time(start)
```

Method (Correct)

```
class Time:
    def print     (self):
        print(f'{self.hour}:{self.minute}:{self.second}')
```

```
>>> start.print     ()
```

Another Example

```
def postpone~~_event~~(e, t):
    e.start.hour += t.hour
    e.start.minute += t.minute
    e.start.second += t.second
```

```
postpone_event(meeting, one_hour)
```

```
meeting.postpone(one_hour)
```

The `__init__` Method

The **`init` method** (short for "initialization") is a **special method** that gets invoked when an object is instantiated.

```
class Time:

  def __init__(self, hour, minute, second):
    self.hour = hour
    self.minute = minute
    self.second = second
```

```
>>> time = Time(9, 45, 20)
>>> time.print()
09:45:20
```

The Advantage of the `__init__` Method

```python
class Time:

  def __init__(self, hour=0, minute=0, second=0):
    self.hour = hour
    self.minute = minute
    self.second = second
```

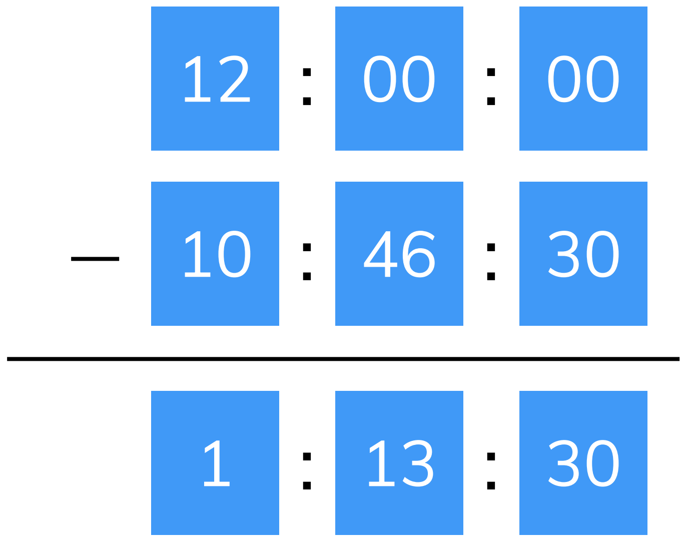
```
>>> time = Time(9, 45)
>>> time.print()
09:45:00
```

The __str__ Method

```
#inside class Time:

  def __str__(self):
    return f'{self.hour}:{self.minute}:{self.second}'
```

```
>>> time = Time(9, 45)
>>> print(time)
09:45:00
```

Operator Overloading



```
noon = Time(12, 0, 0)
now = Time(10, 46, 30)
noon-now
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -:
'Time' and 'Time'
```

http://docs.python.org/3/reference/datamodel.html#specialnames

Operator Overloading: +

```
#inside class Time:

 def __add__(self, other):
   self.hour = self.hour + other.hour
   self.minute = self.minute + other.minute
   self.second = self.second + other.second
   return self
```

```
>>> noon = Time(12, 0)
>>> now = Time(10, 46)
>>> print(noon + now)
22:46:00
```

Exercise

Try to implement “subtract” overload operator

```
#inside class Time:

  def __sub__(self, other):
    # TODO
```

```
>>> noon = Time(12, 0)
>>> now = Time(10, 46)
>>> print(noon - now)
1:14:00
```

thank you!