LECTURER: TAI LE QUY

PROGRAMMING WITH PYTHON

TOPIC OUTLINE

TOPIC OUTLINE

Version Control

6

UNIT 2

CLASSES AND INHERITANCE

STUDY GOALS

- Describe the role and purpose of classes in Python
- Know how to design and develop class objects
- Distinguish between different types of variables' scope
- Understand what is inheritance and how to effectively implement inheritance class hierarchies
- Determine what is method overriding and how to apply it
- Learn how to create iterators and generators

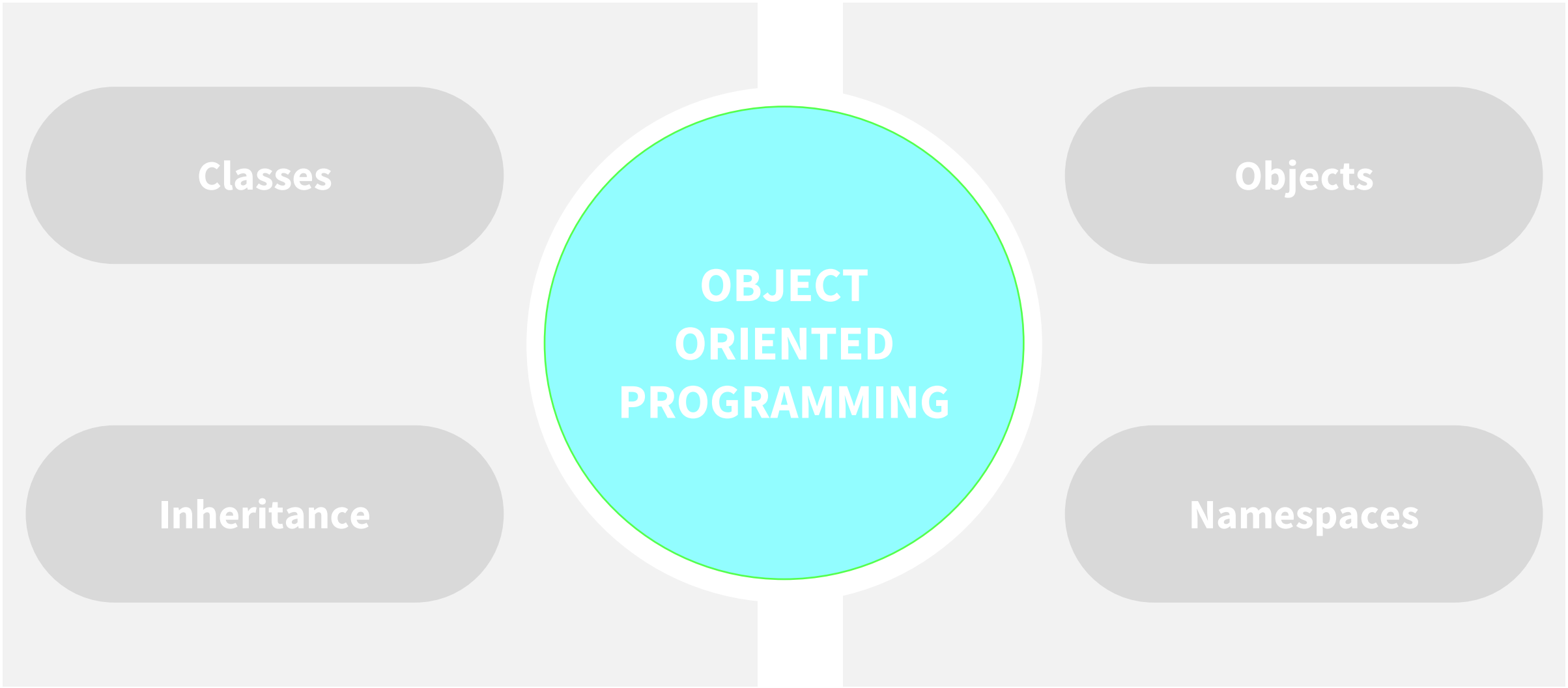
EXPLAIN SIMPLY

1. What is the purpose of namespaces in Python?

2. What is inheritance? How can it improve the quality of software code?

3. What is the role of iterators and generators?

OBJECT ORIENTED PROGRAMMING PARADIGM



Namespaces facilitate well-structured code.
Use to ensure uniqueness of program's names

Namespaces:
a dictionary structure

Keys	Names
Values	Object

Scope

Built-in
built-in functions

Global
imported module definition(s)

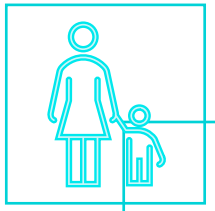
Local
local names inside functions

Scope: part of the program where a name is used without any prefix

- **Local**: local names in current function
- **Enclosing**: a name from the nearest enclosing scope
- **Module**: global names from current module
- **Outermost**: the list of built-in names in the whole program

```
#var1 is in the global namespace
var1 = 5

def function():
    # var2 is in the local namespace
    var2 = 6

    def inner_function():
        # var3 is in the nested local
        # namespace
        var3 = 7
```

class = “blueprint” of a created object



OOP & INHERITANCE

- Code easier to read and to write. Reuse code
- Bound properties and methods to objects
- Encapsulate data; treat code as a black box

object = instance of a class

A class

- created by the <class> keyword
- <self> refers to current instance
- constructor used to initialize an object

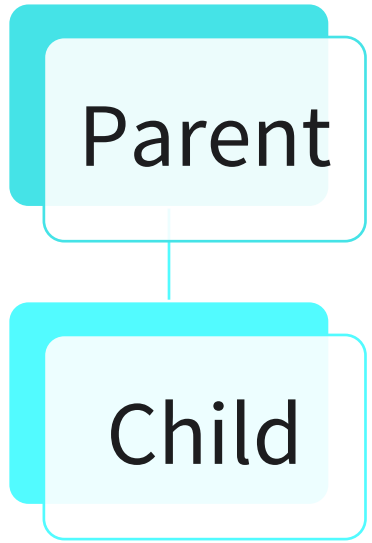
Example

```
class TextBook():

  def __init__(self, pages):
          self.pages = pages

  def print_title(self, title):
          print(title)

  def print_pages(self):
          if self.pages is not None:
            print(self.pages)
```

CLASS INHERITANCE

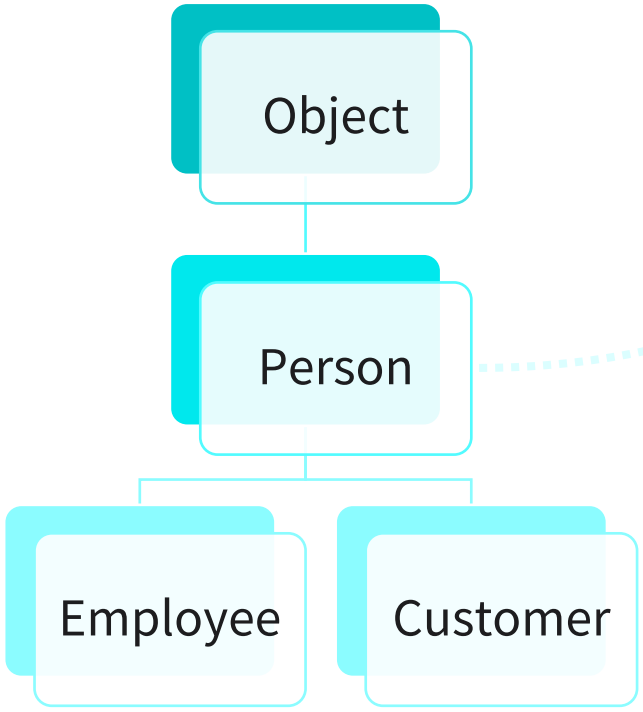
Inheritance: a class inherits methods and properties of another class

- Parent is the class being inherited from
- Child inherits properties and behaviors from parent
- Minimizes duplicate code; improves code organization



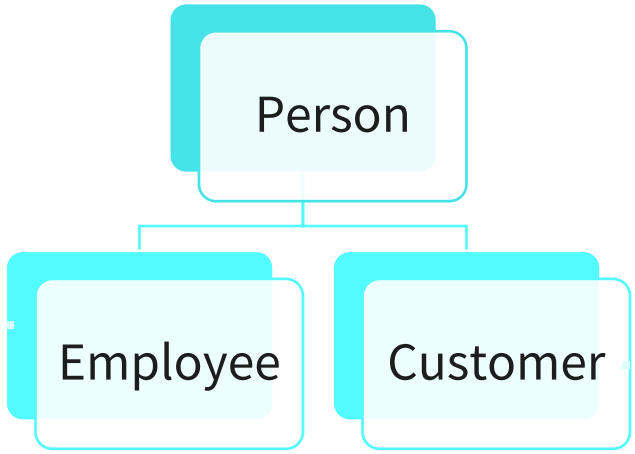
Source of the text: Fabrizio, 2018.

Inheritance Hierarchy Example




```
class Person(object):
  def __init__(self, name):
    self.name = name

  def print_name(self, type):
    print("The " + type + " name is " + self.name)

  def get_name_email (self,email):
    name_email = "Email of " + self.name
                + " is " + email
    return name_email

  def is_employee(self):
    return True
```

CLASS INHERITANCE



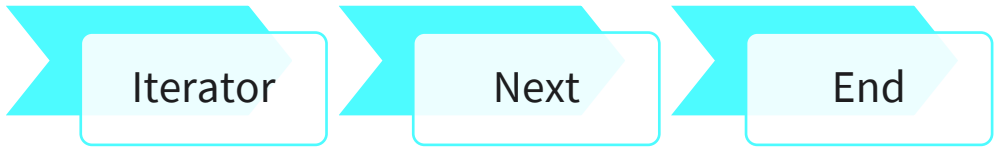

```
class Employee(Person):
  def __init__(self, name):
    super().__init__(name)

  def print_id(self, id):
    print("Employee " + self.name
          + " has id of " + str(id))

  def is_employee(self):
    return True
```

```
class Customer(Person):
  def __init__(self, name):
    super().__init__(name)

  def print_phone(self, phone):
    print("Customer " + self.name   +
          " phone is " + phone)

  def is_employee(self):
    return False
```

Iterable: any object that can be used with a loop

- Built-in iter() function returns an iterator
- Checks and returns the next iterable object

```
def main():
    programming = [“Python”,  “Java”,
                   “R”, “Javascript”]

for item in programming:
        print(item)
```



Generator: creates iterators in an easy way

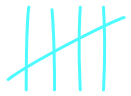
- Implement simple functions, with no return statement using the **<yield>** keyword
- No need to use a customized class with methods __iter()__ and __next()__

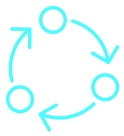
Generator Example

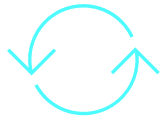
```
def number_generator (low, high):
    while low <= high:
        yield low
        low += 1
```

Module **“itertools”**: efficient built-in iterator for high-speed looping

- Examples include

<count> 

<cycle> 

<repeat> 

Example

```
from itertools import count

def main():
    sequence = count (start=2.5, step = 0.5)
    while (next(sequence)):
        print(next(sequence))
```

Iterator	Arguments	Results	Example
`count()`	start, [step]	start, start+step, start+2*step, …	`count(10) --> 10 11 12 13 14 ...`
`cycle()`	p	p0, p1, … plast, p0, p1, …	`cycle('ABCD') --> A B C D A B C D ...`
`repeat()`	elem [,n]	elem, elem, elem, … endlessly or up to n times	`repeat(10, 3) --> 10 10 10`

REVIEW STUDY GOALS

- Describe the role and purpose of classes in Python
- Know how to design and develop class objects
- Distinguish between different types of variables' scope
- Understand what is inheritance and how to effectively implement inheritance class hierarchies
- Determine what is method overriding and how to apply it
- Learn how to create iterators and generators

UNIT 2

TRANSFER TASK

TRANSFER TASK

Group Work: Exploring Inheritance in Python

READ

Research literature on the topic of inheritance in Python

IDENTIFY

Identify which are the different types of inheritance in Python

EXPLAIN

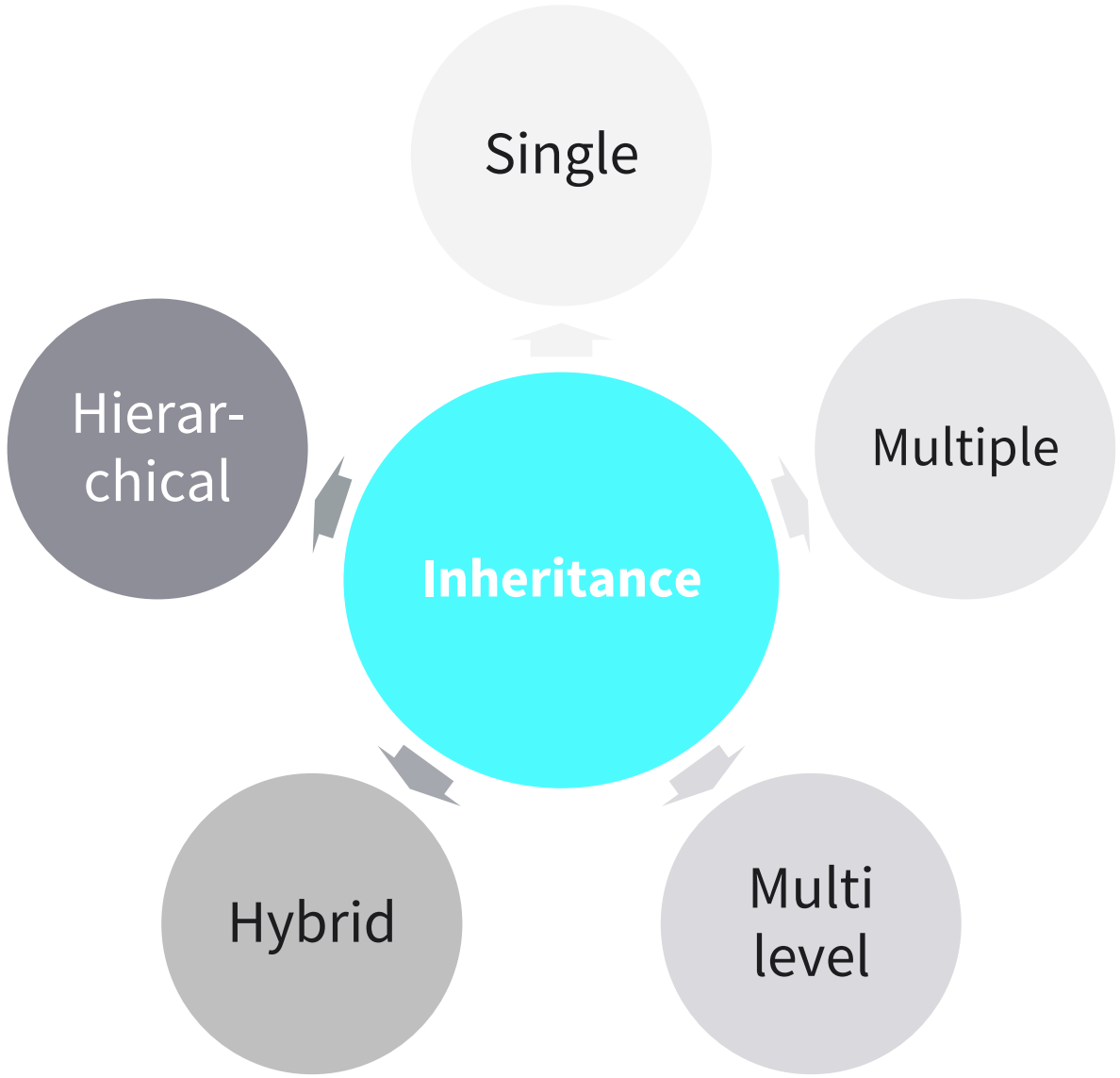
Describe the key characteristics of each type of inheritance

DISCUSS

Discuss your findings and compare them with the other groups

TRANSFER TASK: SAMPLE SOLUTION

- **Single:** a class inherits from another
- **Multiple:** a class inherits from multiple base classes
- **Multilevel:** a class inherits from another which inherits from another
- **Hierarchical:** more than a class inherits from a class
- **Hybrid:** combination of any two kinds



TRANSFER TASK
PRESENTATION OF RESULTS

Please present your results.

The results will be discussed in plenary.

1. Which are the correct definitions of “class” and “object”?
   a) A class is a “blueprint” of a new created object. An object is an instance of a new class
   b) A class is a “blueprint” of a created object. An object is an instance of a class
   c) A class is a “bluecopy” of a created object. An object is a copy of a class
   d) An object is a “blueprint” of a created class. A class is an instance of an object

2. Which is the correct definition of inheritance in Python?
   a) Class inheritance allows computer programmers to create a class that inherits all the methods and properties from another object
   b) Class inheritance allows computer programmers to create a class that inherits all the methods and functions from another class
   c) Class inheritance allows computer programmers to create a class that inherits all the modules and properties from another class
   d) Class inheritance allows computer programmers to create a class that inherits all the methods and properties from another class

3. Iterators are defined as…
   a) …any object that can be used with a “while loop” statement
   b) …any object that can be used with a “for loop statement”
   c) …any object that can be used with a “for loop” or “while loop” statement
   d) …any object that can be used with a “for item” or “while item” statement

LIST OF SOURCES

Fabrizio, R. (2018). *Learn python programming* (2nd ed.). Packt Publishing.