CSIT881
Programming and Data Structures

Class & Object

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Objectives

- Understand the concepts of Class and Object
    - Define class, create object
    - Instance attribute vs Class attribute
    - Instance method
    - Special (dunder) method
    - Static/Class method
- Class inheritance

Class and object

Object instance
0973427
John Smith

Class Student

Is a blueprint for

Object instance
1882845
Mary Wilson

Object instance
0729032
Ye Yang

Class allows us to group data and functionality together.

Class provides a blueprint for creating individual object instances .

Class and object

Class specifies what kind of data an object can hold

```
Class Staff
--------------------------------------------
Staff number
Employment type (Full time/Part time/Casual)
First name
Last name
Date of birth
...
```

```
Class TV_Program
--------------------------------------------
Channel name
Program title
Start time
End time
Category
...
```

Class and object

An object is an instance of a class.

The terms **object** and **instance** are used interchangeably.

Each object instance has its own data values.

```
Staff Object 1
---------------------------
Staff number = 024161
Employment type = Full time
First name = John
Last name = Smith
Date of birth = 14/01/2000
...
```

```
Staff Object 2
---------------------------
Staff number = 952160
Employment type = Casual
First name = Frogory
Last name = Green
Date of birth = 27/04/2001
...
```

Class and object

Each object is a member of a certain class.

```
TV_Program Object 1
-------------------------------------------
Channel name = SBS
Program title = FIFA 2018: Uruguay v Russia
Start time = 25/06/2018 21:00
End time = 26/06/2018 01:15
Category = Sport
...
```

```
TV_Program Object 2
-------------------------------------------
Channel name = ABC
Program title = Bigfoot Family
Start time = 15/03/2021 19:30
End time = 15/03/2021 21:00
Category = Movies
...
```

Instance attribute vs Class attribute

Some information belongs to individual object instance.

Some other information is common to all objects.

Instance attribute: data belongs to individual object instance.

Class attribute: data that is common to all objects.
(Some classes do not have any class attributes.)

Instance attribute vs Class attribute

Instance attribute: data belongs to individual object instance.

For example,

- Each student object has its own first name, last name and student id, etc...
- Each staff object has its own staff id, date of birth, employment type, etc…
- Each TV program object has its own channel name, program title, start time, end time, etc...

Instance attribute vs Class attribute

```
class Student:
#{

  email_domain = "solla.sollew.edu"
  student_dir = "/user/student"

  def __init__(self, id, first_name, last_name):
  #{
    self.id = id
    self.first_name = first_name
    self.last_name = last_name
    ...
  #}

#}
```

class attributes

object/instance attributes

Instance attribute: data belong to individual object instance.

In Python, instance attributes usually get initialised in the special method called **`__init__`**

Instance attribute vs Class attribute

Class attribute: data that is common to all objects.

(Some classes do not have any class attributes.)

For example,

- All students share the same email domain `solla.sollew.edu`
- All students share the same Unix student directory `/user/student`
- All staffs share the cloud work directory `/prv/doc/staff`

Instance attribute vs Class attribute

```
class Student:
#{

  email_domain = "solla.sollew.edu"
  student_dir = "/user/student"

  def __init__(self, id, first_name, last_name):
  #{
    self.id = id
    self.first_name = first_name
    self.last_name = last_name
    ...
  #}

#}
```

class attributes

object/instance attributes

Class attribute: data that is common to all objects.

Instance method vs Class/Static method

Instance method:

- Deal with a particular individual object instance

Static / Class method:

- Do NOT deal with individual object instance
- Common to all object instances

Instance method vs Class/Static method

Instance method:

- Deal with particular individual object instance

For example:

- Get the full name of a Student object
- Get the email address of a Staff object
- Update the title of a TV-Program object
- Update the start time of a TV-Program object

Instance method vs Class/Static method

Instance method:

- Deal with individual object instance attributes
- The first argument (**`self`**) is always referred to the object instance

```
class Student:
#{

  def __init__(self, id, first_name, last_name)...

  def __repr__(self)...

  def fullname(self)...

  def print_detail(self)...

#}
```

Instance method vs Class/Static method

Instance method:

- Deal with individual object instance attributes
- The first argument (**`self`**) is always referred to the object instance

```
class TV_Program:
#{

  def __init__(self, channel, title, start_time,...)...

  def __str__(self)...

  def get_length_in_minutes(self)...

  def time_left_in_minutes(self, reference_time)...

#}
```

Instance method vs Class/Static method

Instance method:

- instance method can be invoked from an object

```
staff2.update_employment_type("Casual")
staff3.update_employment_type("Fulltime")

length2 = tv_program2.get_length_in_minutes()
length5 = tv_program5.get_length_in_minutes()

minute_count5 = tv_program5.time_left_in_minutes(now)
minute_count3 = tv_program3.time_left_in_minutes(now)
```

Instance method vs Class/Static method

Special instance method:

- Some instance methods are called **special** methods, or **dunder** methods.
- Special/dunder methods have the double underscores in the method name

```
class TV_Program:
#{

  def __init__(self, channel, title, start_time,...)...

  def __str__(self)...

  def __repr__(self)...

#}
```

Instance method vs Class/Static method

Static / Class method:

- Do NOT deal with an individual object instance
- Common to all object instances

For example:

- Get the shared student email domain (`solla.sollew.edu`)
- Get the total number of students
- Get the total number of TV programs on a certain channel on a certain day

Instance method vs Class/Static method

Static / Class method:

- static/class method can be invoked from **class name**

```
contact_email = Student.admin_email()

url = Student.uni_website()

studentObj = Student.find_by_student_id("0783122")

tv_program_list1 = TV_Program.find_by_time(now)

tv_program_list2 = TV_Program.find_by_channel("SBS", now)
```

Instance method vs Class/Static method

Class method vs static method:

- The first argument (**`cls`**) of a class method is always referred to the class

```
class Student:
#{
  email_domain = "solla.sollew.edu"
  student_dir = "/user/student"

  @classmethod
  def admin_email(cls):
    return "admin@" + cls.email_domain

  @staticmethod
  def uni_website():
    return "http://www.solla.sollew.edu"

#}
```

Defining class and creating object

```
class Fish:
#{

  def __init__(self, name, color, address):
  #{
    self.name = name
    self.color = color
    self.address = address
  #}

#}
```

object attributes

```
# creating fish objects
shark = Fish("Bruce", "gray", "Sydney aquarium")

goldfish = Fish("Goldie", "orange", "Darling River")

angelfish = Fish("Finley", "blue", "Joe’s fish tank")
```

Defining class and creating object

```
class Student:
#{

  def __init__(self, id, first_name, last_name):
  #{
    self.id = id
    self.first_name = first_name
    self.last_name = last_name
  #}

#}
```

object attributes

```
# creating student objects
student1 = Student("0973427", "John", "Smith")

student2 = Student("1882845", "Mary", "Wilson")

student3 = Student("0729032", "Ye", "Yang")
```

Accessing object instance attributes

```
class Fish:

  def __init__(self, name, color, address):
    ...
```

```
# creating fish objects
shark = Fish("Bruce", "gray", "Sydney aquarium")
goldfish = Fish("Goldie", "orange", "Darling River")
angelfish = Fish("Finley", "blue", "Joe’s fish tank")
```

```
# get object attributes
print(shark.name)
print(shark.color)
print(shark.address)
```

Accessing object instance attributes

```
class Student:

  def __init__(self, id, first_name, last_name):
    ...
```

```
# creating student objects
student1 = Student("0973427", "John", "Smith")
student2 = Student("1882845", "Mary", "Wilson")
student3 = Student("0729032", "Ye", "Yang")
```

```
# get object attributes
print(student1.id)
print(student1.first_name)
print(student1.last_name)
```

Modify object instance attributes

```
angelfish = Fish("Finley", "blue", "Joe’s fish tank")

# display object attributes
print("Before: ")
print(angelfish.name)
print(angelfish.color)
print(angelfish.address)

# change fish address
angelfish.address = "Uni duck pond"

# display object attributes after update
print("After: ")
print(angelfish.name)
print(angelfish.color)
print(angelfish.address)
```

```
Before:
Finley
blue
Joe’s fish tank
```

```
After:
Finley
blue
Uni duck pond
```

Modify object instance attributes

```
student2 = Student("1882845", "Mary", "Wilson")

# display object attributes
print("Before: ")
print(student2.id)
print(student2.first_name)
print(student2.last_name)

# change student last name
student2.last_name = "Davis"

# display object attributes after update
print("After: ")
print(student2.id)
print(student2.first_name)
print(student2.last_name)
```

```
Before:
1882845
Mary
Wilson
```

```
After:
1882845
Mary
Davis
```

Defining an object instance method

```
class Student:
#{
  . . .

  def fullname(self):
  #{

    return self.first_name + " " + self.last_name

  #}

#}
```

Instance method:

- Automatically pass the object instance (**`self`**) as the first parameter

Defining an object instance method

```
class Student:

  def fullname(self):
  #{

    return self.first_name + " " + self.last_name

  #}
```

```
# creating a student object
student1 = Student("0973427", "John", "Smith")
```

```
# calling method - from the object instance
print(student1.fullname())
```

Documenting Python code

```
class Student:
#{
  """
  Class Student represents a student
  """

  def fullname(self):
  #{
    """
    Get student's full name
    """
    return self.first_name + " " + self.last_name
  #}

#}
```

It is important to write documentation of your class and methods.
This helps users to understand the usage and functionality of your code.

```
help(Student)
```

Help method

```
print(help(Student))
```

```
class Student(builtins.object)
 |  Class Student represents a student
 |  with the following attributes:
 |    id: student number
 |    first_name: first name
 |    last_name: last name
 |    username: Unix account username
 |
 |  Methods defined here:
 |
 |  __init__(self, id, first_name, last_name)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __str__(self)
 |      Return str(self).
 |
 |  email(self)
 |      Get student's email: username@domain
 |
 |  email_alias(self)
 |      Get student's friendly-looking email:
 |      firstname.lastname.3IDdigits@domain
 |
 |  fullname(self)
 |      Get student's full name
 |
 |  home_dir(self)
 |      Get student's Unix home directory: studentDir/username
 |
 |  print_detail(self)
 |      Display student detail
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  email_domain = 'solla.sollew.edu'
 |
 |  student_dir = '/user/student'
```

Case study example:

Consider a fictional University called Solla Sollew where

- Each student is given a unique student id (for example, student `John Smith` has student id `0973427`)
- Each student has a username constructed from the first name initial, the last name initial and the first 3 digits of the student id (`John Smith` username is `js097`)
- Student username is constructed at the enrolment day and will **never be changed** even though student may change their name.
- Each student is given a Unix home directory (`John Smith` home directory is `/user/student/js097`)
- Each student is given an email and it will never be changed even though student may change their name (`John Smith` email is `js097@solla.sollew.edu`)
- Each student is given an email alias (`John Smith` email alias is `John.Smith.097@solla.sollew.edu`).
- When student name is changed then this alias also gets changed automatically (for example, if `John Smith` last name changed to `Lee` then his email alias is automatically changed to `John.Lee.097@solla.sollew.edu`)

Defining class Student

```
class Student:
  """
  Class Student represents a student
  with the following attributes:
    id: student number
    first_name: first name
    last_name: last name
    username: Unix account username
  """

  email_domain = "solla.sollew.edu"
  student_dir = "/user/student"

  def __init__(self, id, first_name, last_name):
    ...
```

class attributes

object attributes

Defining class Student

```
class Student:
  def __init__(self, id, first_name, last_name):
    self.id = id
    self.first_name = first_name
    self.last_name = last_name

    # username is constructed in the beginning
    # and will not change if name changed
    # username = lowercase initials + first 3 id digits
    self.username = first_name[0].lower() + last_name[0].lower() + id[0:3]
```

object attributes

```
# creating 3 student objects
student1 = Student("0973427", "John", "Smith")

student2 = Student("1882845", "Mary", "Wilson")

student3 = Student("0729032", "Ye", "Yang")
```

Creating Student objects

```
class Student:

  def __init__(self, id, first_name, last_name):
    ...
```

```
# creating 3 student objects
student1 = Student("0973427", "John", "Smith")

student2 = Student("1882845", "Mary", "Wilson")

student3 = Student("0729032", "Ye", "Yang")
```

Accessing object instance attributes

```
class Student:

  def __init__(self, id, first_name, last_name):
    ...
```

```
# creating 3 student objects
student1 = Student("0973427", "John", "Smith")

student2 = Student("1882845", "Mary", "Wilson")

student3 = Student("0729032", "Ye", "Yang")

# get object attributes
print(student1.id)
print(student1.first_name)
print(student1.last_name)
print(student1.username)
```

Accessing class attributes

```
class Student:

  email_domain = "solla.sollew.edu"
  student_dir = "/user/student"
```

```
# creating 3 student objects
student1 = Student("0973427", "John", "Smith")
student2 = Student("1882845", "Mary", "Wilson")
student3 = Student("0729032", "Ye", "Yang")

# get class attributes
print(Student.email_domain)
print(Student.student_dir)
```

Modify object instance attributes

```
student2 = Student("1882845", "Mary", "Wilson")

# display object attributes
print("Before: ")
print(student2.id)
print(student2.first_name)
print(student2.last_name)

# change student last name
student2.last_name = "Davis"

# display object attributes after update
print("After: ")
print(student2.id)
print(student2.first_name)
print(student2.last_name)
```

```
Before:
1882845
Mary
Wilson
```

```
After:
1882845
Mary
Davis
```

Modify class attributes

```
# change email domain
Student.email_domain = "mail.solla.sollew.edu"

# change student directory
Student.student_dir = "/usr/home/student"
```

Defining an object instance method

```
class Student:

  def fullname(self):
    """
    Get student's full name
    """

    return self.first_name + " " + self.last_name
```

Instance method:

- Automatically pass the object instance (**`self`**) as the first parameter
- May use instance attribute and instance method

Defining an object instance method

```
class Student:

  def fullname(self):
    """
    Get student's full name
    """

    return self.first_name + " " + self.last_name
```

```
# creating a student object
student1 = Student("0973427", "John", "Smith")

# calling method - from the object instance
print(student1.fullname())
```

Defining an object instance method

```
class Student:

  def fullname(self):
    return self.first_name + " " + self.last_name
```

```
# creating a student object
student2 = Student("1882845", "Mary", "Wilson")

# display object attributes
print("Before: ")
print(student2.fullname())

# change student last name
student2.last_name = "Davis"

# display object attributes after update
print("After: ")
print(student2.fullname())
```

```
Before:
Mary Wilson
After:
Mary Davis
```

Defining an object instance method

```
class Student:

  def email(self):
    """
    Get student's email: username@domain
    """
    return self.username + "@" + Student.email_domain
```

```
# creating a student object
student2 = Student("1882845", "Mary", "Wilson")

# display email
print(student2.email())
```

```
mw188@solla.sollew.edu
```

Defining an object instance method

```
class Student:

  def email_alias(self):
    """
    Get student's friendly-looking email:
    firstname.lastname.3IDdigits@domain
    """
    return self.first_name + "." + self.last_name + "." + self.id[0:3] + "@" + Student.email_domain
```

```
# creating a student object
student2 = Student("1882845", "Mary", "Wilson")

# display email alias
print(student2.email_alias())
```

```
Mary.Wilson.188@solla.sollew.edu
```

Defining an object instance method

```
class Student:

  def home_dir(self):
    """
    Get student's Unix home directory:
    studentDir/username
    """
    return Student.student_dir + "/" + self.username
```

```
# creating a student object
student2 = Student("1882845", "Mary", "Wilson")

# display home directory
print(student2.home_dir())
```

```
/user/student/mw188
```

Defining an object instance method

```
class Student:

  def print_detail(self):
    print("Student ID: " + self.id)
    print("First name: " + self.first_name)
    print("Last name: " + self.last_name)
    print("Full name: " + self.fullname())
    print("Username: " + self.username)
    print("Email: " + self.email())
    print("Email alias: " + self.email_alias())
    print("Home directory: " + self.home_dir())
```

```
# creating a student object
student2 = Student("1882845", "Mary", "Wilson")

# display details
student2.print_detail()
```

Defining an object instance method

```
# creating a student object
student2 = Student("1882845", "Mary", "Wilson")

print("Before:")
student2.print_detail()

# change student last name
student2.last_name = "Davis"

print("After:")
student2.print_detail()
```

```
Before:
Student ID: 1882845
First name: Mary
Last name: Wilson
Full name: Mary Wilson
Username: mw188
Email: mw188@solla.sollew.edu
Email alias: Mary.Wilson.188@solla.sollew.edu
Home directory: /user/student/mw188
```

Defining an object instance method

```
# creating a student object
student2 = Student("1882845", "Mary", "Wilson")

print("Before:")
student2.print_detail()

# change student last name
student2.last_name = "Davis"

print("After:")
student2.print_detail()
```

```
After:
Student ID: 1882845
First name: Mary
Last name: Davis
Full name: Mary Davis
Username: mw188
Email: mw188@solla.sollew.edu
Email alias: Mary.Davis.188@solla.sollew.edu
Home directory: /user/student/mw188
```

Special (dunder) method

We have seen a special (dunder) method:

```
class Student:
  def __init__(self, id, first_name, last_name):
```

Now we will write another special (dunder) method:

```
class Student:
  def __str__(self):
```

Why do we need this method `__str__` ?
Try this and see the result:

```
student2 = Student("1882845", "Mary", "Wilson")
print("Object student2 is " + str(student2))
```

```
Object student2 is <__main__.Student object at 0x7f282523ecf8>
```

Special (dunder) method

```
class Student:

  def __str__(self):
    return "{0} ({1})".format(self.fullname(), self.id)
```

Now try this and see the result:

```
student2 = Student("1882845", "Mary", "Wilson")

print("Object student2 is " + str(student2))
```

```
Object student2 is Mary Wilson (1882845)
```

Special (dunder) method

```
class Student:

  def __repr__(self):
    return "Student('{0}', '{1}', '{2}')" \
     .format(self.id, self.first_name, self.last_name)
```

```
student2 = Student("1882845", "Mary", "Wilson")

print(repr(student2))
```

This method gives us the code to construct the object

```
Student('1882845', 'Mary', 'Wilson')
```

Static/Class method

Use the decorator `@staticmethod` to define a static method:

```
class Student:
  @staticmethod
  def uni_website():
    """
    Returns the University website address
    """
    return "http://www.solla.sollew.edu"
```

Use class name to call static method:

```
# calling static method
print("Uni website: " + Student.uni_website())
```

Static/Class method

Use the decorator `@classmethod` to define a static method:

```
class Student:
  @classmethod
  def admin_email(cls):
    """
    Returns the University admin email
    """
    return "admin@" + cls.email_domain
```

In the code above, `cls.email_domain` is the same as `Student.email_domain`

Use class name to call class method:

```
# calling class method
print("Admin email: " + Student.admin_email())
```

Static/Class method

One can easily change a class method to a static method, and vice versa.

```
@classmethod
def admin_email(cls):
    """
    Returns the University admin email
    """
    return "admin@" + cls.email_domain
```

Change class method to static method

```
@staticmethod
def admin_email():
    """
    Returns the University admin email
    """
    return "admin@" + Student.email_domain
```

Static/Class method

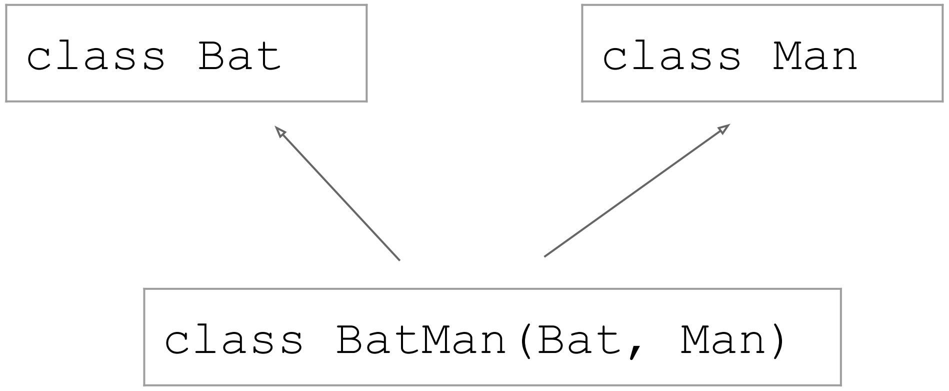
One can easily change a class method to a static method, and vice versa.

```
@staticmethod
def uni_website():
    """
    Returns the University website address
    """
    return "http://www.solla.sollew.edu"
```

Change static method to class method

```
@classmethod
def uni_website(cls):
    """
    Returns the University website address
    """
    return "http://www.solla.sollew.edu"
```

Class inheritance



Python supports **multiple** class inheritance: a child class can inherit from multiple parent classes.

Class inheritance allow child class:

- To inherit all parent attributes and methods;
- To override parent attributes;
- To override parent methods.

Example:

Consider the fictional Solla Sollew University again:

- Each postgraduate student must register a thesis title
- Each postgraduate student is given a Unix home directory in a graduate directory (`Adrian Creedon (0945720)` home directory is `/user/gradstudent/ac094`, if this student was a undergraduate, his home directory would be `/user/student/ac094`)
- Each postgraduate student is given a home page (`Adrian Creedon` home page is `www.solla.sollew.edu/ac094`)

Defining inheritance

```
class PostGradStudent(Student):
  """
  Class PostGradStudent represents a postgraduate student
  """

  student_dir = "/user/poststudent"

  def __init__(self, id, first_name, last_name, thesis):
    # calling parent class constructor
    super().__init__(id, first_name, last_name)

    # initialize thesis title
    self.thesis = thesis
```

overriding class attributes

adding more object attributes

```
# creating 3 postgraduate student objects
pg_student1 = PostGradStudent ("0945720", "Adrian", "Creedon", "Polynomial Approximation of Functions")
pg_student2 = PostGradStudent ("1892418", "Denis", "Carter", "Recursive array constructions")
pg_student3 = PostGradStudent ("0793511", "Kara", "Kaufmann", "On Fundamental Semigroups")
```

Defining inheritance

```
# creating 3 postgraduate student objects
pg_student1 = PostGradStudent ("0945720", "Adrian", "Creedon", "Polynomial Approximation of Functions")
pg_student2 = PostGradStudent ("1892418", "Denis", "Carter", "Recursive array constructions")
pg_student3 = PostGradStudent ("0793511", "Kara", "Kaufmann", "On Fundamental Semigroups")
```

```
# display object attributes

print(pg_student1.id)
print(pg_student1.first_name)
print(pg_student1.last_name)

print(pg_student1.thesis)
```

This is from parent class

This is from child class

Adding attribute and method

```
class PostGradStudent(Student):

  web_domain = "www.solla.sollew.edu"

  def web_address(self):
    """
    Get student's web address:
    webDomain/username
    """
    return PostGradStudent.web_domain + "/" + self.username
```

```
pg_student1 = PostGradStudent ("0945720", "Adrian", "Creedon", "Polynomial Approximation of Functions")

print(pg_student1.web_address())

print(PostGradStudent.web_domain)
```

Overriding method

```python
class Student:
  student_dir = "/user/student"

  def home_dir(self):
    """
    Get student's Unix home directory:
    studentDir/username
    """
    return Student.student_dir + "/" + self.username
```

```python
class PostGradStudent(Student):
  student_dir = "/user/poststudent"

  def home_dir(self):
    """
    Get student's Unix home directory: studentDir/username
    Override the parent method with a new directory
    """
    return PostGradStudent.student_dir + "/" + self.username
```

Overriding method

```
# creating 3 student objects
student1 = Student("0973427", "John", "Smith")
student2 = Student("1882845", "Mary", "Wilson")
student3 = Student("0729032", "Ye", "Yang")
```

```
# creating 3 postgraduate student objects
pg_student1 = PostGradStudent ("0945720", "Adrian", "Creedon", "Polynomial Approximation of Functions")
pg_student2 = PostGradStudent ("1892418", "Denis", "Carter", "Recursive array constructions")
pg_student3 = PostGradStudent ("0793511", "Kara", "Kaufmann", "On Fundamental Semigroups")
```

```
# compare the home directory between 2 students
print(student1.home_dir())

print(pg_student1.home_dir())
```

```
/user/student/js097
/user/poststudent/ac094
```

Overriding method

```
class PostGradStudent(Student):

  def print_detail(self):
    """
    Display student detail
    """
    super().print_detail()

    print("Thesis: " + self.thesis)
    print("Web address: " + self.web_address())
```

calling from parent class

additional info from child class

```
pg_student1 = PostGradStudent ("0945720", "Adrian", "Creedon", "Polynomial Approximation of Functions")
pg_student1.print_detail()
```

```
Student ID: 0945720
First name: Adrian
Last name: Creedon
Full name: Adrian Creedon
Username: ac094
Email: ac094@solla.sollew.edu
Email alias: Adrian.Creedon.094@solla.sollew.edu
Home directory: /user/poststudent/ac094
Thesis: Polynomial Approximation of Functions
Web address: www.solla.sollew.edu/ac094
```