This course supports the assessment for Scripting and Programming - Foundations. The course covers 6 competencies and represents 3 competency units.

Introduction

Overview

This course provides an introduction to programming covering data structures, algorithms, and programming paradigms. The course presents you with the concept of an object as well as the object-oriented paradigm and its importance. A survey of languages is covered and the distinction between interpreted and compiled languages is introduced.

Getting Started

Welcome to Scripting and Programming - Foundations! This course is designed to help those who have never coded before learn what it means to script and program, and to build basic coding skills. You will be using Python language for coding in this course, but the skills you learn will be necessary in learning *any* programming language. You will be using learning resources from Udacity, Lynda.com, and Codecademy to complete your studies. Each activity in this course is designed to be completed in a single learning session that takes around 1-2 hours. Competency in this course will be demonstrated by the successful completion of an objective assessment.

Watch the following Getting Started video for this course:

Note: To download this video, right-click the following link and choose "Save as...": download video.

Competencies

This course provides guidance to help you demonstrate the following 6 competencies:

- **Competency 4015.2.1: Introduction to Computer Programming**
  The graduate performs basic computer programming including working with data types, constants, variables, operator types, expressions, and functions.
- **Competency 4015.2.2: Basic Constructs of Programming**
  The graduate implements basic constructs of programming, including working with control structures.
- **Competency 4015.2.3: Object-oriented Concepts**
  The graduate integrates the object-oriented programming paradigm in scripting and programming.
- **Competency 4015.2.4: Algorithms**
  The graduate analyzes algorithms, including algorithm efficiency, and recursion.
- **Competency 4015.2.6: The Design Process**

The graduate describes steps of the design process.

- **Competency 4015.02.7: Programming Languages**
  The graduate compares various programming languages.

Preparing for Success

The information in this section is provided to detail the resources available for you to use as you complete this course.

Learning Resources

- Udacity - CS 101 Introduction to Computer Science
- Lynda.com videos

Optional Learning Resources

These are additional coding exercises to practice with Python. You will need to create a free account with Codecademy to access their resource and save your progress.

To sign up, follow these steps:

- Go to Learn Python in Codecademy
- Click "Sign Up."
- Click "Sign up with Google."
- If requested, enter your wgu.edu email address.

Course Instructor Support

Course instructors are able to share tips on approaches, tools, and skills that can help you apply the content you are studying. They also provide guidance in assessment preparation strategies and can help you troubleshoot areas of deficiency.

- Email your course instructors: cmweb@wgu.edu
- Schedule a course instructor appointment

Course Instructor Tips for Success

- This course uses version 2.7.3 of Python. Be careful with any outside resource that may use another version, as the syntax can be different.
- The best way to learn a programming language is to actually use or type the code while you are using the learning resources (e.g., watching the Udacity videos). Learning programming is best done by doing! You will be using built-in development tools in Udacity and Codecademy to complete all of the coding exercises. There is no need to download your own interpreter, but if you prefer to do so you can download and run PyCharm on your computer.
- Details are extremely important in programming! To ensure that you have identified the details throughout the course, it is crucial that you take notes. If you find it difficult to take thorough notes when watching videos, make use of the text alternatives. Udacity has course notes for each lesson and Lynda.com has a full transcript of each video.

Pacing Guide

Accelerated Pacing

If you would like to complete the course in fewer than 6 weeks, please use this interactive pacing planner. You determine how many days per week you will study and how many weeks you want to spend on the course and the pacing planner will create a weekly checklist to help you meet your goals.

Standard Pacing

The standard pacing guide suggests a weekly structure to pace your completion of learning activities. It is provided as a suggestion and does not represent a mandatory schedule. Follow the pacing guide carefully to complete the course in the suggested timeframe.

Week 1

- Introduction to Programming
- Object-Oriented Analysis, Design, and Use Cases

Week 2

- Getting Started with Python
- Operators and Looping with Python

Week 3

- Boolean Logic
- Built-in Objects in Python

Week 4

- How Python Programs Run
- Recursion and Advanced Looping with Python

Week 5

- Choosing a Programming Language
- Terms Quiz
- Pre-assessment

Week 6

- Review coaching report and study
- Complete the objective assessment

Note: This pacing guide does not replace the course. Please continue to refer to the course for a comprehensive list of the resources and activities.

Coursework

Follow the course of study and check off each activity as you complete it.

Introduction to Programming

This topic highlights the following objectives:

- Identify the basic principles of programming.
- Define the concept of a class.
- Distinguish the three principles of object-oriented programming.

What is Programming?

What is programming? What does a programmer *do*? What is a programming language? How do I get started if I want to be a programmer?

Learn the answers to these questions and more from Simon Allardice in Foundations of Programming: Fundamentals from Lynda.com. Once you are in the Lynda.com course, you can continue right on through the introduction and Lesson 1, or you can come back to the direct links that follow:

- Introduction
  - Simon's Welcome, then click the Get Started box to complete a quick check of what you already know.
  - Making the most of this course.
- Lesson 1: Programming Basics
  - What is programming?
  - What is a programming language?
  - Writing source code.
  - Compiled and interpreted languages.

Object-Oriented Programming

Some of the most in-demand and widely-used programming languages are *object-oriented*. But what does this mean? And what implications does it have for the approach you will be taking, as a programmer?

Watch as Simon Allardice addresses these questions in Foundations of Programming: Object-Oriented Design from Lynda.com. *Note*: the "Introduction" and "Lesson 1" you previously completed were from a different Lynda.com course: this is not a repeat, but rather new content that you need to watch. Once you are in the Lynda.com course, you can continue right on through the introduction and Lesson 1, or you can come back to the direct links that follow:

- Introduction
  - Simon's Welcome, then click the Get Started box to complete a quick check of what you already know.
  - Who this course is for.
  - What to expect from this course.
  - Exploring object-oriented analysis, design, and development.
  - Reviewing software development methodologies.

- Lesson 1: Core Concepts
  - Why we use object-orientation.
  - What is an object?
  - What is a class?
  - What is abstraction?
  - What is encapsulation?
  - What is inheritance?
  - What is polymorphism?

Classes in Python

Throughout this course we will be exploring programming concepts using the Python language. At this point Python coding probably looks like a foreign language to you, and you may be wondering how we work with classes in Python. We can create a class like this:

```
class ():
```

The word class is required, followed by whatever we wish to name our class. Within the parentheses we tell the class where it should inherit its properties. On the lines following the colon we would define whatever properties and methods this particular class needs to contain. Properties are attributes that the object will have: methods are functions that the object can call upon.

As an example, this class is called *childClass* and inherits from a class named *parentClass* and has a property defined for the color blue. This class also has a method called jumpMethod:

```
class childClass(parentClass):
    color = 'blue'

    def jumpMethod(self):

        print 'jump'
```

Don’t worry if the coding isn’t entirely clear at this point!

Study Guide

Complete the Introduction to Programming Study Guide created by your course instructor team, then compare your answers with those provided upon submission.

Object-Oriented Analysis, Design, and Use Cases

This topic highlights the following objectives:

- Define the analysis and design process.
- Define UML and explain its use.
- Define class diagrams.
- Describe uses, use cases, actors, and scenarios.

Planning and Diagrams

How do you match objects and classes with the real-world needs and goals of a computer program? With UML—the theory that underlies how your program is going to work.

Follow along carefully as Simon Allardice explains how this works in Foundations of Programming: Object-Oriented Design from Lynda.com. Once you are in the Lynda.com course, you can continue right on through Lessons 2, 3, and 5 or you can come back to the direct links that follow:

- Lesson 2: Object-Oriented Analysis and Design
  - Understanding the object-oriented analysis and design processes.
  - Defining requirements.
  - Introduction to the Unified Modeling Language (UML).
- Lesson 3: Utilizing Use Cases
  - Understanding use cases.
  - Identifying the actors.
  - Identifying the scenarios.
  - Diagramming use cases.
  - Employing user stories.
- Lesson 5: Creating Classes
  - Creating class diagrams.
  - Converting class diagrams to code.

Study Guide

Reflect on what you learned from Simon Allardice in the videos. Coming up next in this course, you'll watch the process of designing and programming a web search engine. What are the requirements of a search engine? What use cases can you think of?

Complete the Object-Oriented Analysis, Design, and Use Cases Study Guide created by your course instructor team, then compare your answers with those provided upon submission.

Getting Started with Python

This topic highlights the following objectives:

- Explain the concept and grammar of a programming expression.
- Manipulate the value of a variable.

- Define the concepts of a string data type and string operations.
- Verify operations with strings.

Programming: How to Get Started

Now that you have a sense of the theoretical foundation of a computer program, it's time to start programming. Watch the following video for an overview of how to successfully leverage the learning resources that will help you learn to program:

Note: To download this video, right-click the following link and choose "Save as...": download video.

Just a few minutes after starting this video, you will have written your first program! Follow along carefully and complete each quiz and activity presented along the way. Watch the Udacity video How to Get Started. You can use the Notes to supplement the video.

Study Guide

Complete the Getting Started with Python Study Guide created by your course instructor team, then compare your answers with those provided upon submission.

Operators and Looping with Python

This topic highlights the following objectives:

- Use the correct Python operator.
- Describe comparison programming expressions.
- Implement a While loop.
- Define the concept of a function.
- Define algorithms.

How to Repeat

Learn

Watch the Udacity video How to Repeat. You can use the Notes to supplement the video.

Operations

Learn

Dave Evans introduced several operators in lessons 1 and 2, and you'll see additional operators in future lessons. You should be familiar with the operators covered in Udacity's Complete Python Reference Guide under the first section, 1.1 Arithmetic Expressions.

Additionally, when multiple mathematics operations are present within one expression (example: 5 + 2 * 1) Python follows the standard mathematical order of operations. This is also called operator precedence and it follows the following sequence.

1. First, perform calculations within parentheses.
2. Next, work left to right and perform all exponentiation
3. Then, work left to right and perform all multiplication and division.
4. Finally, work left to right and perform all addition and subtraction.

A great way to remember this is the phrase "Please Excuse My Dear Aunt Sally," with the first letter of each word a reminder of the order:

- P = parentheses
- E = exponents
- M = multiplication
- D = division
- A = addition
- S = subtraction

Look at the following practical example of a mathematical equation and the steps for resolution:

```
mysum = 5 * 2 + (3 * 4)
```

First, the expression within the parentheses is evaluated:

```
mysum = 5 * 2 + (12)
```

Then, the multiplication is performed:

```
mysum = 10 + (12)
```

Finally, the addition of the two values is calculated:

```
mysum = 22
```

Study Guide

Complete the Operators and Looping with Python Study Guide created by your course instructor team, then compare your answers with those provided upon submission.

Boolean Operators

This topic highlights the following objectives:

- Compose a two-way decision.
- Solve Boolean algebra operations.
- Evaluate Boolean code.

Logical Operators

Boolean Operators

When you are working with the NOT, AND, and OR evaluators, the important property they

have which is different from other operators is that the second operand expression is evaluated only when necessary.

NOT Evaluator

The Logical NOT Operator is used to reverse the logical state of its operand. For example, if a condition is evaluated to false then the Logical NOT operator will make the answer true. Here are a few examples:

not True ? False

not False ? True

not (True and False) ? True

not (True or False) ? False

In the third and fourth example, the parenthetical will be evaluated first so (True and False) will evaluate to false because both operands must be true to evaluate to true. Applying the logical NOT will evaluate the opposite of false which makes the final answer true.

In the last example, the same thought process applies, however, (True or False) evaluates to true due to the "or"; only one operand has to be true for the entire condition to be true. Then applying the logical NOT makes the final answer False.

AND Evaluator

When working with the AND evaluator, if the first expression evaluates to false, then the second expression is ignored and the entire statement evaluates to false. Here are a few examples:

false AND true ? false

(false AND true) and true ? false

(10 < 5) AND (5 < 10) ? false

In both examples the first expression is false so the second expressions, which are true, are never evaluated.

If the first expression evaluates to true then the second expression is also evaluated. If it also evaluates to true then the entire expression evaluates to true. If the second expression evaluates to false then the entire statement is false. Here are a few examples:

true AND true ? true

true AND false ? false

(4<= 10) AND (3 >1) ? true

(5 == 5) AND (3 < 1) ? false

OR Evaluator

When working with the OR evaluator, if the first expression is true, then the second expression is ignored. Here are a few examples:

true OR false ? true

(true OR false) or false ? true

(5 > 2) OR (4 == 4) ? true

If the first expression is false then the entire statement is the value of the second expression. Here are a few examples:

false OR true ? true

false OR false ? false

(5 < 4) OR (5 == 5) ? true

(2 <= 1) OR (8 >=9) ? false

If Statements

Recall Dave's introduction to if statements in Udacity's Lesson 2. Construct your *If* statements carefully! The *If* statement must accurately describe all cases except those that should be excluded, and that can mean using the Boolean operators you just read about. Anything not evaluated as true in your *if* statement triggers the *Else*_condition.

Study Guide

Complete the Boolean Operators Study Guide created by your course instructor team, then compare your answers with those provided upon submission.

Built-in Objects in Python

This topic highlights the following objectives:

- Define the concept of a list in the Python programming language.
- Find elements in a list.
- Use the pop method to remove an item at a given position of a list.
- Implement a For loop.
- Analyze various loop patterns.

How to Manage Data

Watch the Udacity video How to Manage Data. You can use the Notes to supplement the video.

Removing Elements at a Given Position

As you learned in the uDacity videos, we can use .pop() to remove the last element from a list. For example, take a look at the following code:

```
mylist = [5, 2, 3, 1, 4]

mylist.pop()
```

The new value of mylist will now be:

```
mylist = [5, 2, 3, 1]
```

We can also use the .pop() method to specify a specific item within the list to be removed by indicating its index position. Remember that index positions start at 0. If I wanted to use the .pop() method to remove the first element from my list, I would do so like this:

```
mylist = [5, 2, 3, 1, 4]

mylist.pop(0)
```

After running this code the new value of mylist will now be:

```
mylist = [2, 3, 1, 4]
```

Study Guide

Complete the Built-in Objects in Python Study Guide created by your course instructor team, then compare your answers with those provided upon submission.

How Python Programs Run

This topic highlights the following objectives:

- Identify the concept of algorithm cost.
- Identify techniques for measuring speed.
- Explain the relationship between index and speed.
- Implement strategies to increase processing efficiency.

How Programs Run

Watch the Udacity video How Programs Run. Lesson 4 is not needed in order to continue. You can use the Notes to supplement the video.

Study Guide

Complete the How Python Programs Run Study Guide created by your course instructor team,

then compare your answers with those provided upon submission.

Recursion and Circular Definitions

This topic highlights the following objectives:

- Analyze recursion and cases for its use.
- Define circular definitions.

How to Have Infinite Power

Watch the Udacity video How to Have Infinite Power. You can use the Notes to supplement the video.

Explaining Recursion

As you learned in Lesson 6, recursion is a way for a function to call itself. It is a very complex idea, so it may be useful to look further at an example and how each of the parts fit together within the recursive definition. Here is what happens when we run a recursive definition:

1. The recursive procedure is asked to run.
2. If the base case is met, then the procedure will end.
3. If the base case is not met the recursion will begin working and continue until it can reach that base case.

Knowing this, the recursive definition must include two parts: the base case and the recursive case. Within the recursive definition it must also change its state and work towards the base case.

In the Udacity videos they showed an example of a recursive factorial procedure. Watch the following video for further explanation: Reviewing Recursion

Note: To download this video, right-click the following link and choose "Save as...": download video.

Study Guide

Complete the Recursion and Advanced Looping with Python Study Guide created by your course instructor team, then compare your answers with those provided upon submission.

Choosing a Programming Language

This topic highlights the following objectives:

- Analyze different programming languages' suitability for solving different problems.
- Define the concept of a programming library.
- Describe the role of a library in selecting a programming language.

Exploring the Languages

Learn

By this point, you are a Python programmer! Congratulations! The skills you've built up in Python will help you with whatever programming language you might work with in the future, but they are all different. What are these differences? How do you know the best programming language to use for a particular project?

Follow along carefully as we return now to Simon Allardice for an explanation in Foundations of Programming: Fundamentals from Lynda.com. Watch all of Lesson 14:

- Lesson 14: Exploring the Languages
    - Introduction to languages.
    - C-based languages.
    - The Java world.
    - .NET languages: C# and Visual Basic .NET.
    - Ruby.
    - Python.
    - Objective-C.
    - Libraries and frameworks.

Study Guide

Complete the Choosing a Programming Language Study Guide created by your course instructor team, then compare your answers with those provided upon submission.

Coding Practice

To learn the basics of Python, it is important to practice with the language. Udacity has introduced you to the basic code, but Codecademy is a great tool for additional practice. You will need to create a free account to access the exercises.

To sign up, follow these steps:

- Go to http://www.codecademy.com/tracks/python
- Click "Sign Up."
- Click "Sign up with Google."
- If requested, enter your wgu.edu email address.

Codecademy has a series of interactive exercises that build on knowledge as you progress. **You will be required to finish one lesson before you proceed to the next.** For example, if you attempt to access Lesson 2 it will be locked until you first complete Lesson 1. At that time, you can begin Lesson 2.

Below are the lessons that most closely align with the previous topics in the course, but you will need to complete all previous lessons to progress.

Getting Started with Python:

- Lesson 1: Steps 1 – 12
- Lesson 2

Operators and Looping:

- Lesson 3
- Lesson 4: Steps 1 – 6
- Lesson 8: Steps 1 – 13

Built-In Objects

- Lesson 5: Steps 1 – 8

Classes in Python

- Lesson 11

If you have trouble completing the code, compare your code to the completed code for each step.

Terms Quiz

It's extremely important to understand the terms in this course.

You should be able to complete the quiz below in "Quiz" mode with a score of 90% before taking the pre-assessment.

Terms Quiz

Note: The Quizlet test has fill-in-the-blank/written answers by default. The tool counts spelling, exact wording, and case when computing a score. Feel free to remove those types of questions! You can customize your quiz using the box in the upper right-hand corner.

Pre-assessment

Now that you have completed the courseware, take the pre-assessment (located under the Assessment tab). Use the results of your coaching report in conjunction with the study guides you completed for each topic to master any areas of concern.

Course instructors can offer assistance if you are struggling with any of the questions.

Final Steps

Congratulations on completing the activities in this course! You are now prepared to complete the associated assessment. If you have not already been directed to complete it, schedule and complete the assessment now.