Courses
Forum
Events (2)
Sharing (10+)
Videos
Resources
Exercises
Pages
Archives
Chat Room
My Account
My Assessments
Sign Out

Part 1: Study Guide for PY109 Exam

This assessment will test your knowledge of PY101 and the Introduction to Programming with Python book. It has a huge surface area in that it covers the Python programming language broadly. It does not cover Intermediate Python or Object Oriented Programming.

Specific Topics of Interest

In general, you should be familiar with Python syntax and operators. You should also be able to clearly explain, talk about, or demonstrate the following topics:

- naming conventions: legal vs. idiomatic, illegal vs. non-idiomatic
- type coercions: explicit (e.g., using `int()`, `str()`) and implicit
- numbers
- strings
- f-strings
- string methods
  - `capitalize`, `swapcase`, `upper`, `lower`
  - `isalpha`, `isdigit`, `isalnum`, `islower`, `isupper`, `isspace`
  - `strip`, `rstrip`, `lstrip`, `replace`
  - `split`, `find`, `rfind`

- boolean vs. truthiness
- `None`
- ranges
- list and dictionary syntax
- list methods: `len(list)`, `list.append()`, `list.pop()`, `list.reverse()`
- dictionary methods: `dict.keys()`, `dict.values()`, `dict.items()`, `dict.get()`
- slicing (strings, lists, tuples)
- operators
  - Arithmetic: `+`, `-`, `*`, `/`, `//`, `%`, `**`
  - String operators: `+`
  - List operators: `+`
  - Comparison: `==`, `!=`, `<`, `>`, `<=`, `>=`
  - Logical: `and`, `or`, `not`
  - Identity: `is`, `is not`
  - operator precedence

- mutability and immutability
- variables
  - naming conventions
  - initialization, assignment, and reassignment
  - scope
  - `global` keyword
  - variables as pointers
  - variable shadowing

- conditionals and loops
  - `for`
  - `while`

- `print()` and `input()`
- exceptions (when they will occur and how to handle them)
- Functions:
  - definitions and calls
  - return values
  - parameters vs. arguments
  - nested functions
  - output vs. return values, side effects

- expressions and statements

Using a REPL

> Python REPLs that run code as you type it do not run that code the same way as when you run the code from a `.py` file. This can lead to discrepancies in the behavior of your code. That can influence your answers on the assessment.
>
> Unless otherwise stated, all assessment questions that use code examples or that expect you to write code assume that the code should be run from a `.py` file. **We strongly advise against testing your code with a REPL.**

How to Answer the Assessment Questions

The questions in this assessment will typically test your knowledge and understanding at more than one level.

- On one level, the question will test your ability to parse code and to describe it with precision, or test your knowledge of some specific syntactical aspect or language-specific feature of the Python programming language.
- On another level, the question will check your understanding of some deeper underlying principle; this might be some more fundamental aspect of the Python language or a non-language-specific programming concept.

When answering the questions, you should:

- Explain your reasoning with reference to specific lines in the program. You can use line numbers to refer to particular lines of code where necessary.
- Answer with precision. For example, say "function definition"or "function invocation" as opposed to just "function" when the distinction is important.
- Highlight any specific syntactical conventions or technical observations where relevant.
- Identify the fundamental concept or concepts demonstrated by the question.

Example

Examine the code example below. The last line outputs the string `'Hello'` rather than the string `'Hi'`. Explain what is happening here and identify the underlying principle that this demonstrates.

Copy Code

```
1  greeting = 'Hello'
2
3  def greet():
4      greeting = 'Hi'
5      return greeting
6
7  greet()
8  print(greeting)
```

Compare the following possible answers to this question:

A) `greeting` is set to `'Hello'` on line 1. `greeting` is set to `'Hi'` on line 4. Line 8 outputs `greeting`, which is `'Hello'`.

B) Inside the function `greet`, a new local variable `greeting` is created and set to `'Hi'`. This does not affect the global variable `greeting`, which remains `'Hello'`. This is why line 8 outputs `'Hello'`.

C) The code defines a global variable `greeting` with the value `'Hello'`. In the `greet` function, a new local variable, also named `greeting`, is assigned `'Hi'`. This local variable is separate from the global one. This is why the `print` invocation on line 8 outputs the unchanged value of the global variable, `'Hello'`.

D) The global variable `greeting` is initially set to `'Hello'`. In the `greet` function, when `greeting` is assigned `'Hi'`, it creates a local variable separate from the global `greeting`. Therefore, the `print` invocation on line 8 outputs the value of the global `greeting`, which is `'Hello'`. This code demonstrates Python's variable scope rules, specifically highlighting how variables defined in the global scope cannot be reassigned within a function's local scope without using the `global` keyword.

While none of these answers is technically incorrect, they all answer the question with varying degrees of detail and precision.

- Answer **A** describes what is happening in the code example but does so in a basic way using imprecise language. This response wouldn't be sufficient to receive full points for any of the questions in the assessment.
- Answer **B** again describes what is happening but with a more detailed explanation. This answer would score higher than answer A but might still not suffice to receive full points, as it misses a deeper understanding of the example's principles.
- Answer **C** explicitly mentions that the `greeting` variable within the `greet` function is a local variable separate from the global variable `greeting` initialized on line 1. The student would probably lose a point here for not identifying the underlying principle.
- Answer **D** goes a step further than **C** by explaining why this is important and the underlying principle it demonstrates, i.e., the fact that Python has particular scoping rules that affect a variable's accessibility, specifically that global variables can't be reassigned within the local function scope unless `global` keyword is used. This answer would receive full points in an actual assessment.

Bullet Points

Many students attempt to use bullet points to answer the questions on the exam. This makes sense in some situations:

- You have a list of explicit reasons why some code does what it does.
- You have a list of pros and cons.
- You want to provide a list of things.

In short, they work well for **lists**. (Notice that we used a bullet list to list this list of lists!) However, they don't always work as complete answers for a question. You don't speak in bullet lists; don't write with lists.

To illustrate, consider the following hypothetical explanation of the example code from the previous section:

- Line 1 initializes a variable named `greeting` to a string `'Hello'`.
- On lines `3-5` function `greet` is defined.
- Line 4 assigns local variable `greeting` to `'Hi'`.
- Line 5 returns `greeting` variable.
- Line 7 prints the value of `greeting` to the console.
- Variables defined in an outer scope can't be reassigning within the inner function scope without using the `global` keyword.

This answer is essentially a *laundry list* of facts about the code. Unfortunately, laundry lists aren't very effective as answers on the assessment. They are difficult to follow, and often leave it to the reader to piece together the logic behind the list.

In the above list, for instance, there's no logical progression that actually explains what is happening. Instead, the student has simply listed a bunch of facts about each line of code, plus one item that talks about scope. However, a program is not a series of independent lines of code. Code depends on what happened before, and it influences what happens later. There's nothing in the laundry list that connects those individual bits of code together.

From the grader's point of view, this answer is incomplete:

- it doesn't mention that global variable `greeting` can't be reassigned within a function `greet`.
- it doesn't talk about the fact that `greeting` on line 4 is not the same variable as the one shown on lines 1 and 7.
- It doesn't tie the statement about variables to the other statements.

In short, it leaves the grader with the burden of tying your bullet points together in a coherent whole.

These faults can be addressed, to a degree, in a bullet point answer. However, the laundry list approach often leads students to overlook these missing details. Paragraphs make it easier to think about the bigger picture since you're striving for clarity, not a list of everything you can think of.

Some students overcompensate by listing a bunch of impertinent facts about the code instead of focusing on the question. For instance, consider the following question and code:

> What does this code print and why?

Copy Code

```
1 def replace(string, value):
2     while True:
3         break
4
5     string = value
6
7 greet = 'Hey!'
8 replace(greet, 'Hello')
9 print(greet)
```

A student might list several irrelevant facts about this code:

- `while` is a loop that might execute forever.
- `break` causes the loop to end immediately.
- `'Hello'` and `'Hey'` are strings.

None of that information is untrue. However, it is mostly clutter for the grader for this particular question. It would be fine to provide these details if the question is looking for a line-by-line explanation, but that's not what it is asking. You may also lose points if the extra detail says something wrong. Saying something like "The loop has no effect on this code" would be reasonable, though it's not really an important aspect of this code, so could be skipped.

Instead, focus on what the code prints and why it prints that. In particular, the pertinent issues here are that the code prints `Hey!` since the `replace` function doesn't mutate the string passed in as its first argument; it can't mutate the string since strings are immutable. On line 5, we reassign the 2nd argument to the `string` variable. However, reassignment of a variable never mutates the value it contains, so it has no effect on the string contained by `greet`.

In general, a clearly written paragraph is easier to understand and grade than a laundry list. While we won't penalize you for using bullet points, it's important to realize that bullet points have weaknesses that are difficult to see when you're writing.

Precision of Language

Most questions require that you explain some code with words. It's important to be able to describe why something happens using precise vocabulary and be able to pinpoint the principle (scope, shadowing, etc.) at work. In other words, be precise and don't be vague.

For example, let's take the following piece of code.

Copy Code

```
1 hello = "Hello, world!"
2
3 def my_func():
4     print(hello)
5
6 my_func()
```

If asked to describe what the `my_func` code does, you might be tempted to say:

> The result of the function is `"Hello, world!"`.

This statement isn't wrong, but, it's imprecise and doesn't help us understand the function. If you had written that as an answer, you might score a 2/5 on the question (40% is not a passing score).

A more precise answer would be something along the lines of this:

> The function outputs `Hello, world!`, which it obtains from the global variable `hello`, then returns `None`. Functions in Python have access to variables defined in the outer scope.

In programming, we're always concerned with the output and the return value, as well as any object mutation and non-local variables being used. We need to speak in those terms, and not use vague words like "results."

When writing answers to the test questions, be as precise as possible, and use the proper vocabulary. Doing this will help you debug and understand more complex code later in your journey. If your definitions are not precise, you won't be able to lean on them to decompose complicated code. Also, you will likely not be able to pass this assessment.

Some Specifics

For the purposes of this assessment, we will use some terms in very precise ways. You should be extremely precise in the language that you use as well. Doing so will prevent misunderstandings during grading. Relying on precise language will help both you and us understand each other.

These areas are outlined below.

Assignments

Consider the following assignment:

Copy Code

```
greeting = 'Hello'
```

Most of the Launch School material describes this assignment as:

> The `greeting` variable is assigned to the string `'Hello'`.

However, there are places where we describe this code as:

> The string `'Hello'` is assigned to the `greeting` variable.

Both of these are acceptable in the assessment. Try to be consistent though to avoid confusion.

Variables

Unless mentioned specifically, we use the term **variable** in the broadest sense possible. On this exam, that means that all of the following should be treated as variables:

- Variables and constants
- Function names
- Function parameters

Note in particular that dictionary key names **are not** variables, nor are the elements of a collection.

Truthiness

In the assessment, we want you to be very clear about the distinction between *truthy* and *falsy* values and the boolean values `True` and `False`.

In Python, the *falsy* values include `False`, `0`, `0.0`, `None`, `""`, and empty collections like `[]`, `()` and `{}`. All other values, the *truthy* values, are said to *evaluate to true*. Note that saying that a value evaluates to true or false is **not** the same as saying that those values **are** `True` or `False`, or that they are **equal to** `True` or `False`. These may seem like subtle distinctions, but they are important ones.

Suppose we ask you to describe what is happening in this code:

Copy Code

```
a = "Hello"

if a:
    print("Hello is truthy")
else:
    print("Hello is falsy")
```

If your answer says that "`a` is `True`" or that "`a` is equal to `True`" in the conditional on line 3, we would likely deduct points from your score. A much better answer would say that "`a` is truthy" or that "`a` evaluates to true" on line 3.

To sum up:

- Use "evaluates to true" or "is truthy" when discussing expressions that only have to be truthy.
- Use "evaluates to false" or "is falsy" when discussing expressions that only have to be falsy.
- Do not use "is `True`" or "is equal to `True`" unless you are specifically discussing the boolean value `True`.
- Do not use "is `False`" or "is equal to `False`" unless you are specifically discussing the boolean value `False`.

Distinctions

Be clear about the following distinctions:

- Parameters are the names assigned to a function's arguments; arguments are the values that get passed to the function.
- Variables are not passed to or returned by functions: **references to objects** are passed.
- Truthiness vs Boolean values (see above)

Time Management

There are approximately 9-10 questions on this exam, though that may vary a bit. Be sure to practice time management to ensure you don't go over the time limit. For example, with 10 questions in a 2-hour exam, you will need an average of 12 minutes for every question. If you spend 15 minutes per question, you will rapidly become pressed for time.

Also, take some time at the beginning of the exam to quickly review all of the questions so you can see which ones are going to require more time than the easier questions. This will give a better feel for how much time you can **really** devote to each question.

Finally, don't forget to leave a little time to review your answers. In particular, make sure you answered all parts of every question. Many students forget to answer all parts of each question, and some of them end up losing a significant number of points.

A few of our students have written articles on time management on the JS109 and RB109 exams. Even though these articles weren't written for PY109, they are worth a read:

- How to prepare for Written Assessments and Manage your time while taking them!
- Why I Ran Out of Time Taking Launch School's RB109 Written Assessment (and what I'm changing so it doesn't happen again)

> The above articles were written for some older versions of the RB109 and JS109 exams. Those exams had 3 hour time limits and more questions than the exam for this assessment. These discrepancies don't change the usefulness of those articles and their relevance to PY109.

Online Resources

When an online resource conflicts with Launch School materials, the Launch School materials should be used on the assessment. We can't grade assessments using information that differs from what we present, especially when that information may be incorrect.

Additional Tips

When writing code for an assessment question, run your code often to check it.

Some Launch School students have blogged about their assessment experiences:

- In the video How I Study and Prepare For Launch School's Assessments, Felicia describe's how she prepares for Launch School assessments, both exams and interviews. This is an excellent video with advice that almost everybody can benefit from.
- In the article Belling the Cat: Practical Lessons in Passing the PY109 ssessment, Herun talks about his process of passing the PY109 assessment. There is no actual Python code in this article, but it's chock full of valuable tips for being prepared for the assessment.
- Marilyn has some great advice about What You Don't Need to Pass RB109. While her article is about the Ruby RB109 exam, her advice is also valuable for the Python PY109 exam.
- Zach had a rough time making his way through the first two Launch School assessments. In I Failed Programming 101 Three Times, he describes his struggle and frustration trying to pass these assessments, and how he eventually embraced the Launch School way.
- Raúl talks about his preparation and experiences as a non-native English speaker in this interesting article.
- Drew shares a few tips to set you up for success and to make your Launch School journey a little less daunting.
- Dalibor wrote a blog article about his strategy for success on Launch School's written assessments.
- Callie's blog article has a wealth of useful information about preparing for both parts of the assessment: the exam and the interview. Callie speaks specifically of the Ruby-based RB109 assessments, but her advice and recommendations apply equally as well to PY109.

> The above articles were written for some older versions of the RB109 and JS109 exams. Those exams had 3 hour time limits and more questions than the exam for this assessment. These discrepancies don't change the usefulness of those articles.

These articles are well worth your time; don't pass them up! We hope to see some PY109-centric articles soon!

> You marked this topic or exercise as completed.