

DSE5002 Homework/Lab 01

These problems are all in the textbook, I just cut and pasted the questions into this Python Notebook

In future homeworks, you will have to construct the notebook yourself, I'm trying to go easy on you this week.

Exercises from

<https://allendowney.github.io/ThinkPython/chap01.html>

and

<https://allendowney.github.io/ThinkPython/chap02.html>

Run these examples in an AI LLM model (chatbot, Claude, etc, take your pick)

Briefly comment in the cell below on how well this worked- only a few sentences are needed.

1.9.1. Ask a virtual assistant

As you work through this book, there are several ways you can use a virtual assistant or chatbot to help you learn.

If you want to learn more about a topic in the chapter, or anything is unclear, you can ask for an explanation.

If you are having a hard time with any of the exercises, you can ask for help.

In each chapter, I'll suggest exercises you can do with a virtual assistant, but I encourage you to try things on your own and see what works for you.

Here are some topics you could ask a virtual assistant about:

Earlier I mentioned bitwise operators but I didn't explain why the value of $7 \wedge 2$ is 5. Try asking "What are the bitwise operators in Python?" or "What is the value of 7 XOR 2?"

I also mentioned the order of operations. For more details, ask "What is the order of operations in Python?"

The round function, which we used to round a floating-point number to the nearest integer, can take a second argument. Try asking "What are the arguments of the round function?" or "How do I round pi off to three decimal places?"

There's one more arithmetic operator I didn't mention; try asking "What is the modulus operator in Python?"

Most virtual assistants know about Python, so they answer questions like this pretty reliably. But remember that these tools make mistakes. If you get code from a chatbot, test it!

I wanted to experiment with ChatGPT o3-mini-high, which has been impressive when compared to the traditional instruct models. I asked it a more abstract question about python to see how it would respond. Question and response, below:

Question

How will the removal of the GIL in python version 3.13 affect multithreading performance? Will python ever have performance that rivals a compiled language?

Response

The removal of the GIL in Python 3.13 is expected to unlock true parallelism for CPU-bound threads, which means that programs that perform heavy computation concurrently can potentially run more efficiently on multi-core systems. However, the benefits and limitations should be understood in context:

Multithreading Performance Gains

- **True Parallel Execution:** Without the GIL, threads in CPU-bound tasks will be able to execute concurrently on multiple cores, which can lead to significant performance improvements in scenarios that previously suffered from serialized execution.
- **Increased Overhead:** To ensure thread safety without the GIL, the interpreter will need to adopt more fine-grained locking and synchronization mechanisms. This may introduce additional overhead in some cases, possibly affecting single-threaded performance or increasing the complexity of thread management.
- **Impact on I/O-Bound Workloads:** For many I/O-bound applications, the impact of removing the GIL might be less pronounced because these applications typically spend much of their time waiting for external operations (where the GIL is less of a bottleneck).

Compatibility and Ecosystem Considerations

- **C Extensions:** Many third-party C extensions have been written with the assumption of the GIL's protection. These extensions will need to be revised for thread safety in a no-GIL environment, potentially creating transitional challenges.
- **Legacy Code:** Some existing code might need adjustments to fully benefit from the new concurrency model, meaning the gains could be gradual as the ecosystem adapts.

Comparing Python with Compiled Languages

- **Inherent Overhead:** Python is a dynamically typed, interpreted language, which means that even with improvements in multithreading, it carries overhead that is absent in statically compiled languages like C or C++. For raw, compute-intensive tasks, this overhead means Python will generally remain slower.
- **Performance via Extensions:** In practice, many performance-critical parts of Python applications already rely on optimized C libraries (or alternatives like Cython, Numba, or PyPy). These strategies allow Python to achieve near-compiled performance in specific domains without changing the core language.
- **Design Philosophy:** Python's strengths lie in developer productivity, ease of use, and a rich ecosystem. While improvements like GIL removal can make a notable difference for concurrent applications, they don't transform the language into a full substitute for compiled languages in terms of raw speed.

Future Outlook

The removal of the GIL is an important milestone in Python's evolution, particularly for applications that benefit from parallel CPU execution. However, the language's dynamic nature and interpreter design mean that—without a radical overhaul of its core architecture—it is unlikely to consistently rival the performance of compiled languages on compute-heavy tasks. Instead, Python will continue to excel in its niche by leveraging interoperability with high-performance libraries and specialized tools when needed.

In summary, while Python 3.13's move toward a no-GIL environment promises improved multithreading performance in specific scenarios, it doesn't signal a complete convergence with the raw speed of compiled languages. The improvements will enhance Python's concurrency capabilities, but the language's fundamental design still favors flexibility and ease of use over low-level, high-speed computation.

1.9.3. Exercise

When you learn about a new feature, you should try it out and make mistakes on purpose. That way, you learn the error messages, and when you see them again, you will know what they mean. It is better to make mistakes now and deliberately than later and accidentally.

You can use a minus sign to make a negative number like -2. What happens if you put a plus sign before a number? What about 2++2?

What happens if you have two values with no operator between them, like 4 2?

If you call a function like round(42.5), what happens if you leave out one or both paren

{Write the code that checks all of these }theses?

```
In [15]: case_1 = +2
print(case_1)
```

2

This is actually something that I did not know about python, which I have been programming in for about 2.5 years. Turns out, python implicitly handles the polarity of positive numbers. I asked ChatGPT o3-mini-high about this to learn more:

Question

Does python implicitly handle the polarity of numbers? I.E. `+2` is evaluated as `2`?

Response

Yes, Python treats `+2` the same as `2`. The unary plus operator (`+`) simply returns the value of its operand unchanged. This means that writing `+2` is just a more explicit way to denote the positive number 2, and the interpreter evaluates it as 2.

For example:

```
print(+2) # Output: 2
```

In this case, the unary plus does not alter the value—it simply confirms that the number is positive.

```
In [25]: case_2 = 4 2
print(case_2)
```

```
Cell In[25], line 1
case_2 = 4 2
^
```

```
SyntaxError: invalid syntax
```

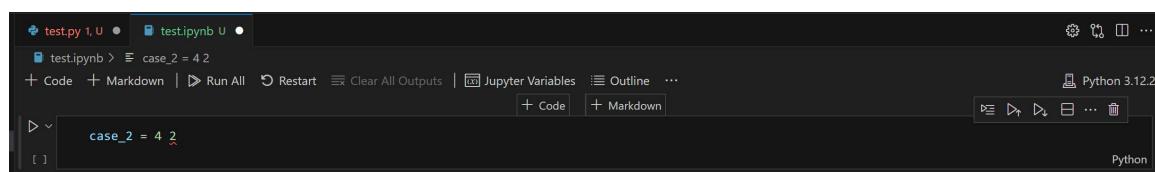
This is evaluated as a syntax error at execution time, as the space indicates to the interpreter that the integer "4" has terminated, and it now expects an operator prior to the integer 2.

The interpreter implicitly removes additional whitespace when the proper sequence of int - operator - int or float - operator - float is met. See below:

```
In [27]: case_2_alt =    4    +    2
print(case_2_alt)
```

6

Interestingly, jupyter notebooks that run via the jupyter launcher in the anaconda navigator don't have a linter by default. I typically use VS Code for my IDE, and it handles debugging in real time with a debugging extension.



It seems like I can install debugging capabilities in the conda environment as well, so I will have to do some research.

```
In [37]: case_3 = round(42.5
```

```
Cell In[37], line 1
case_3 = round(42.5
```

```
^
SyntaxError: incomplete input
```

Again, this is another syntax error, as the function is missing its terminating parenthesis. This would be the same for strings missing the terminating quotes.

1.9.4. Exercise Recall that every expression has a value, every value has a type, and we can use the type function to find the type of any value.

What is the type of the value of the following expressions? Make your best guess for each one, and then use type to find out.

765 - int

2.718 - float

'2 pi' - str

abs(-7) - int (this is a nested function, so it will be evaluated from inner-most to outer-most. The abs() function returns a float or int, depending on what the input was)

abs(-7.0) - float (this is a nested function, so it will be evaluated from inner-most to outer-most. The abs() function returns a float or int, depending on what the input was)

abs - built_in_function

int - probably class... not sure though

type - probably class... not sure though

{write the code below to check these}

```
In [41]: type(765)
```

```
Out[41]: int
```

```
In [43]: type(2.718)
```

```
Out[43]: float
```

```
In [45]: type('2 pi')
```

```
Out[45]: str
```

```
In [47]: type(abs(-7))
```

```
Out[47]: int
```

```
In [49]: type(abs(-7.0))
```

```
Out[49]: float
```

```
In [51]: type(abs)
```

```
Out[51]: builtin_function_or_method
```

```
In [53]: type(int)
```

```
Out[53]: type
```

```
In [55]: type(type)
```

```
Out[55]: type
```

I was off on the last two... so I asked ChatGPT, because I have been scratching my head about this.

Question

Why does `type(int)` in python return type and not class?

Response

In Python, classes are first-class objects created by a metaclass. The built-in metaclass is called `type`, which means every class (including `int`) is an instance of `type`. When you evaluate `type(int)`, you're asking for the type of the object `int`, and since `int` is a class created by the `type` metaclass, it returns `type`.

To illustrate:

```
print(type(2)) # Output: <class 'int'>
print(type(int)) # Output: <class 'type'>
```

Here, `2` is an instance of the class `int`, so its type is `<class 'int'>`. On the other hand, `int` itself is a class, and because classes in Python are objects created by the metaclass `type`, `type(int)` returns `<class 'type'>`.

This design underscores one of Python's core philosophies: everything is an object, and classes themselves are objects that are instances of a metaclass (by default, `type`).

I was completely unaware of the metaclass/subclass structure of python, but it makes sense if the design goal of the language is to have dynamic, implicitly determined types. The block,

below, is the behavior I was expecting, but I now understand that int is a subclass of the metaclass type.

```
In [61]: print(int)
```

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

1.9.5. Exercise The following questions give you a chance to practice writing arithmetic expressions.

How many seconds are there in 42 minutes 42 seconds?

How many miles are there in 10 kilometers? Hint: there are 1.61 kilometers in a mile.

If you run a 10 kilometer race in 42 minutes 42 seconds, what is your average pace in seconds per mile?

What is your average pace in minutes and seconds per mile?

What is your average sps per hour?eed in mile

{Write your s per hour?

```
In [63]: print(f'{(42*60)+42} seconds')
```

```
2562 seconds
```

```
In [67]: print(f'{round(10/1.61, 2)} miles')
```

```
6.21 miles
```

```
In [73]: seconds = (42*60)+42
miles = (10/1.61)
pace = seconds/miles

print(f'Pace: {pace} seconds per mile')
```

```
Pace: 412.482 seconds per mile
```

```
In [83]: from math import floor
```

```
In [87]: print(f'Pace: {floor(pace/60)} minutes and {floor(pace%60)} seconds per mile')
```

```
Pace: 6 minutes and 52 seconds per mile
```

```
In [95]: print(f'Average speed: {round(3600/pace, 2)} miles per hour')
```

```
Average speed: 8.73 miles per hour
```

2.11.2. Exercise Repeating my advice from the previous chapter, whenever you learn a new feature, you should make errors on purpose to see what goes wrong.

We've seen that `n = 17` is legal. What about `17 = n`?

How about `x = y = 1`?

In some languages every statement ends with a semi-colon (`;`). What happens if you put a semi-colon at the end of a Python statement?

What if you put a period at the end of a statement?

What happens if you spell the name of a module wrong and try to import `maath`?

In [121...]

```
17=n
```

```
Cell In[121], line 2
17var=n
^
SyntaxError: invalid decimal literal
```

In [125...]

```
17var=n
```

```
Cell In[125], line 1
17var=n
^
SyntaxError: invalid decimal literal
```

In [123...]

```
var17=n
```

```
NameError                                                 Traceback (most recent call last)
Cell In[123], line 1
----> 1 var17=n

NameError: name 'n' is not defined
```

In [131...]

```
n=17
var17=n
print(n, var17)
```

17 17

Variable definitions can never begin with integers in python. Integers can be contained within the variable name, however.

In [133...]

```
x=y=1

print(x)
print(y)
```

1
1

This is a handy trick for initializing several variables. Often, I include flags in my programs to pass state changes from parent classes to child classes, and I will initialize all the flags to 0 in this format.

```
In [109...]
string = 'Hello World!';
integer = 4+2;
print(string)
print(integer)
```

Hello World!

6

This is my rebuttal to the embedded systems folks at my company that program in C/C++ and hate the formatting of python :).

```
In [113...]
string = 'Hello World!'.
integer = 4+2.
```

```
Cell In[113], line 1
  string = 'Hello World!'.
```

SyntaxError: invalid syntax

Periods are probably more strictly typed than semi-colons... I have a feeling that the interpreter will always try to coerce a period into a string, whereas the semi-colon can be coerced into whitespace or stripped from the line entirely.

```
In [115...]
from maath import floor
```

```
ModuleNotFoundError
```

Traceback (most recent call last)

```
Cell In[115], line 1
```

```
----> 1 from maath import floor
```

ModuleNotFoundError: No module named 'maath'

Imports are pulled from packages that need to be available in the global or virtual environment for the kernel. The 'import' keyword tells the interpreter to look in the environment (usually under env/Lib/site-packages) for the module name that you are importing. If it cannot find this package, it will return an error.

2.11.3. Exercise Practice using the Python interpreter as a calculator:

Part 1. The volume of a sphere with radii $4/3(\pi r^3)$

. What is the volume of a sphere with radius 5? Start with a variable named radius and then assign the result to a variable named volume. Display the result. Add comments to indicate that radius is in centimeters and volume in cubic centimeters.

Part 2. A rule of trigonometry says that for any value of θ , $\sin^2 \theta + \cos^2 \theta = 1$. Let's see if it's true for a specific value of like 42.

Create a variable named `x` with this value. Then use `math.cos` and `math.sin` to compute the sine and cosine of `x`, and the sum of their squared.

The result should be close to 1. It might not be exactly 1 because floating-point arithmetic is not exact—it is only approximately correct.

Part 3. In addition to `pi`, the other variable defined in the `math` module is `e`, which represents the base of the natural logarithm, written in math notation as e . If you are not familiar with this value, ask a virtual assistant “What is `math.e`?” Now let’s compute three ways:

Use `math.e` and the exponentiation operator `(**)`.

Use `math.pow` to raise `math.e` to the power 2.

Use `math.exp`, which takes as an argument a value, `x`, and computes e^x .

You might notice that the last result is slightly different from the other two. See if you can find out which is correct.

```
In [146...]: from math import pi
```

```
In [152...]: radius=5  
volume=(4/3)*pi*(radius**3)
```

```
print(f'The volume of a sphere with radius {radius} centimeters is: {round(volume,2)}
```

The volume of a sphere with radius 5 centimeters is: 523.6 cm³

```
In [154...]: from math import cos, sin
```

```
In [170...]: x=42  
print(f'Cosine of x: {round(cos(x),2)}\t\tSine of x: {round(sin(x), 2)}\t\tRSS: {co
```

Cosine of x: -0.4

Sine of x: -0.92

RSS: 1.0

```
In [178...]: from math import e, pow, exp
```

```
In [180...]: print(f'Method 1: {e**2}\tMethod 2: {pow(e, 2)}\tMethod 3: {exp(2)}')
```

Method 1: 7.3890560989306495
Method 2: 7.3890560989306495
Method 3: 7.38905609893065

The output from [Wolfram Alpha](#) is:

7.389056098930650227230427460575007813180315570551847324087127822522573796
which indicates that Method 3 is correct.

