

Start time: 6:41 PM, 10/2/24

Objective: run through exercises on Exercism to refresh myself on Python

The screenshot displays the Exercism web interface for the Python exercise "Guido's Gorgeous Lasagna". The left pane shows the code editor with a Python file named `lasagna.py`. The code includes constants for `EXPECTED_BAKE_TIME` (40 minutes) and `TIME_PER_LAYER` (2 minutes), and functions for `bake_time_remaining`, `preparation_time_in_minutes`, `elapsed_time_in_minutes`, and `total_time_in_minutes`. The right pane shows the "Results" tab, indicating that all tasks have been passed. A congratulatory message states: "Sweet. Looks like you've solved the exercise! Good job! You can continue to improve your code or, if you're done, submit an iteration to get automated feedback and optionally request mentoring." Below this, a list of tasks is shown, all marked as completed:

- TASK 1 Define expected bake time in minutes
- TASK 2 Calculate remaining bake time in minutes
- TASK 3 Calculate preparation time in minutes
- TASK 4 Calculate total elapsed cooking time (prep + bake) in minutes
- TASK 5 Update the recipe with notes

At the bottom of the code editor, there are buttons for "Stuck? Get help", "Run Tests", and "Submit".

Taken at 7:12 PM (this took a while to figure out what the exercise was asking for)

← Back to Exercise Python / Ghost Gobble Arcade Game

```
arcade_game.py
1  """Functions for implementing the rules of the classic arcade game Pac-Man."""
2
3
4  def eat_ghost(power_pellet_active, touching_ghost):
5      """Verify that Pac-Man can eat a ghost if he is empowered by a power pellet.
6
7      :param power_pellet_active: bool - does the player have an active power pellet?
8      :param touching_ghost: bool - is the player touching a ghost?
9      :return: bool - can a ghost be eaten?
10     """
11
12     return power_pellet_active and touching_ghost
13
14
15 def score(touching_power_pellet, touching_dot):
16     """Verify that Pac-Man has scored when a power pellet or dot has been eaten.
17
18     :param touching_power_pellet: bool - is the player touching a power pellet?
19     :param touching_dot: bool - is the player touching a dot?
20     :return: bool - has the player scored or not?
21     """
22
23     return touching_power_pellet or touching_dot
24
25
26 def lose(power_pellet_active, touching_ghost):
27     """Trigger the game loop to end (GAME OVER) when Pac-Man touches a ghost without his power pellet.
28
29     :param power_pellet_active: bool - does the player have an active power pellet?
30     :param touching_ghost: bool - is the player touching a ghost?
31     :return: bool - has the player lost the game?
32     """
33
34     return touching_ghost and not power_pellet_active
35
36
37 def win(has_eaten_all_dots, power_pellet_active, touching_ghost):
38     """Trigger the victory event when all dots have been eaten.
39
40     :param has_eaten_all_dots: bool - has the player "eaten" all the dots?
41     :param power_pellet_active: bool - does the player have an active power pellet?
42     :param touching_ghost: bool - is the player touching a ghost?
43     :return: bool - has the player won the game?
44     """
45
46     return has_eaten_all_dots and not lose(power_pellet_active, touching_ghost)
47
```

Stuck? Get help Run Tests Submit

Instructions Results ChatGPT Get help

ALL TASKS PASSED

Sweet. Looks like you've solved the exercise!
Good job! You can continue to improve your code or, if you're done, submit an iteration to get automated feedback and optionally request mentoring.

Submit

TASK 1 Define if Pac-Man eats a ghost +

TASK 2 Define if Pac-Man scores +

TASK 3 Define if Pac-Man loses +

TASK 4 Define if Pac-Man wins +

RT Clear Search 7:29 PM 10/2/2024

Taken at 7:29 PM

← Back to Exercise Python / Grains

```
grains.py
1  def square(number):
2      if not isinstance(number, int):
3          raise ValueError("square must be between 1 and 64")
4      else:
5          if (number > 64) or (number < 1):
6              raise ValueError("square must be between 1 and 64")
7          else:
8              return 2 ** (number - 1)
9
10
11 def total():
12     return square(64) * 2 - 1
13
```

Stuck? Get help Run Tests Submit

Instructions Tests Results ChatGPT Get help

ALL TESTS PASSED

Sweet. Looks like you've solved the exercise!
Good job! You can continue to improve your code or, if you're done, submit an iteration to get automated feedback and optionally request mentoring.

Submit

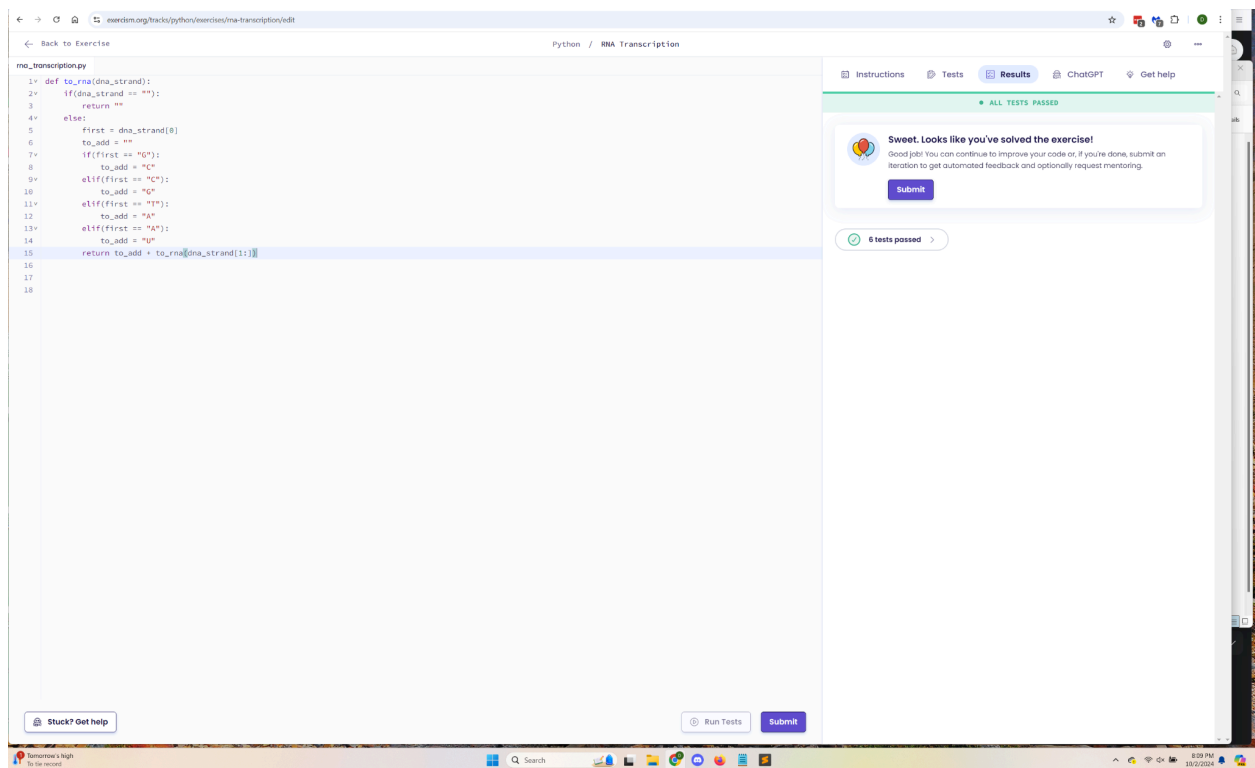
11 tests passed >

Tomorrow's High So far record Search 7:38 PM 10/2/2024

Taken at 7:37 PM



Taken at 8:00 PM (briefly attempted a different exercise, but found the instructions unclear. After that, this exercise was easy)



Taken at 8:09 PM

Back to Exercise

Python / All Your Base

all_your_base.py

```
1< def rebase(input_base, digits, output_base):
2<     if(input_base < 2):
3<         raise ValueError("input base must be >= 2")
4<     if(output_base < 2):
5<         raise ValueError("output base must be >= 2")
6<     toBaseTen = 0
7<
8<     for iter in range(len(digits)):
9<         indexValue = digits[iter]
10<
11<         if(indexValue >= input_base):
12<             raise ValueError("all digits must satisfy 0 <= d < input base")
13<
14<         toBaseTen += input_base ** indexValue * indexValue
15<
16<     newBase = toBaseTen
17<
18<     toStr = str(newBase)
19<     output = []
20<     for i in range(len(toStr)):
21<         output += [toStr[i]]
22<
23<     return output
24<
25<
26< def toNewBase(value, output_base):
27<     output = ""
28<
29<     while(value > 0):
30<         output += value % output_base
31<         value = value / output_base;
32<
33<     return int(output)
34<
```

Stuck? Get help

Run Tests

Submit

Instructions

Tests

Results

ChatGPT

Get help

13 TEST FAILURES

8 tests passed

13 tests failed

Test 1

AllYourBase > single bit one to decimal

CODE RUN

self.assertEqual(rebase(2, [1], 10), [1])

TEST FAILURE

TypeError: can only concatenate str (not "int") to str

Test 2

AllYourBase > binary to single decimal

Test 3

AllYourBase > single decimal to binary

Test 4

AllYourBase > binary to multiple decimal

Test 5

AllYourBase > decimal to binary

Test 6

AllYourBase > trinary to hexadecimal

Test 7

AllYourBase > hexadecimal to trinary

Test 8

AllYourBase > 15 bit integer

Test 9

AllYourBase > empty list

Test 10

AllYourBase > single zero

Test 11

AllYourBase > multiple zeros

Test 12

AllYourBase > leading zeros

Taken at 8:27 PM (Exercise not completed yet)

Back to Exercise

Python / All Your Base

all_your_base.py

```
1< def rebase(input_base, digits, output_base):
2<     if(input_base < 2):
3<         raise ValueError("input base must be >= 2")
4<     if(output_base < 2):
5<         raise ValueError("output base must be >= 2")
6<
7<     if(input_base == output_base):
8<         return digits
9<
10<     if(input_base == 10):
11<         for iter in range(len(digits)):
12<             # TODO: unfinished
13<
14<     elif(output_base == 10):
15<         # TODO: unfinished
16<
17<     else:
18<         return rebase(10, rebase(input_base, digits, 10), output_base)
19<
20<
21<
22< def toNewBase(value, output_base):
23<     output = ""
24<
25<     while(value > 0):
26<         output += value % output_base
27<         value = value / output_base;
28<
29<     return int(output)
30<
```

Stuck? Get help

Run Tests

Submit

Instructions

Tests

Results

ChatGPT

Get help

Introduction

You've just been hired as professor of mathematics. Your first week went well, but something is off in your second week. The problem is that every answer given by your students is wrong! Luckily, your math skills have allowed you to identify the problem: the student answers are correct, but they're all in base 2 (binary)! Amazingly, it turns out that each week, the students use a different base. To help you quickly verify the student answers, you'll be building a tool to translate between bases.

Instructions

Convert a sequence of digits in one base, representing a number, into a sequence of digits in another base, representing the same number.

Note

Try to implement the conversion yourself. Do not use something else to perform the conversion for you.

About Positional Notation

In positional notation, a number in base **b** can be understood as a linear combination of powers of **b**.

The number 42, in base 10 means:

$$(4 \times 10^1) + (2 \times 10^0)$$

The number 101010, in base 2 means:

$$(1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

The number 1020, in base 3 means:

$$(1 \times 3^3) + (1 \times 3^2) + (2 \times 3^1) + (0 \times 3^0)$$

Yes. Those three numbers above are exactly the same. Congratulations!

Exception messages

Sometimes it is necessary to **raise an exception**. When you do this, you should always include a **meaningful error message** to indicate what the source of the error is. This makes your code more readable and helps significantly with debugging. For situations where you know that the error source will be a certain type, you can choose to raise one of the **built in error types**, but should still include a meaningful message.

This particular exercise requires that you use the **raise statement** to "throw" a **ValueError** for different input and output bases. The tests will only pass if you both **raise the exception** and include a meaningful message with it.

To raise a **ValueError** with a message, write the message as an argument to the **exception** type:

```
# for input:
raise ValueError("input base must be >= 2")

# another example for input:
raise ValueError("all digits must satisfy 0 <= d < input base")

# or, for output:
```

Taken at 8:46 PM (Exercise not completed yet)