

Function Ladder

Seven rungs. Build each function in `ladder.py`, run it, check it against its test chip.

Name: _____

Date: _____

Work solo or paired. Every function lives in one Spyder file: `ladder.py`. Run the file with F5, call the function in the console, read what it shows. Your teacher checks your tracker at the end.

Part 1 Climb the ladder (rungs 1–3)

1 Rung 1: no input. Type this in `ladder.py`, run it (F5), then call `lucky()` in the console.

```
def lucky():  
    return 7
```

`lucky()` expected: 7

What did YOU get? _____ ☐ it matches

2 Rung 2: one input. Finish the blank, run it, call `add_one(5)`.

```
def add_one(x):  
    return _____ # x and 1?
```

`add_one(5)` expected: 6

What did YOU get? _____ ☐ it matches

(your turn) Now write your own one-input function in `ladder.py`: any name, any one-input maths.

Name of your function: _____

_____ (_____) expected: _____

What did YOU get? _____ ☐ it matches

3 Rung 3: what is return FOR? This function has no `return`. Add it to the file, run it, call `hello()`.

```
def hello():  
    greeting = "hi" # no return here
```

code	what I think happens	what actually happened	✓
<code>add_one(5)</code>	Example: the console shows 6.	Example: it showed 6.	✓
<code>hello()</code>			

// `return` sends a value back to the console. No return, nothing comes back.

`add_one(10)` expected: 11

What did YOU get? _____ ☐ it matches

Part 2 Rungs 4–5: two inputs, then your four

4 Rung 4: two inputs (worked example). Run it, call `average(3, 5)`.

```
def average(a, b):  
    return (a + b) / 2
```

average(3, 5) expected: 4.0

What did YOU get? _____ ☐ it matches

(your turn) Write `product(a, b)` : it gives back `a` times `b` . Code it in `ladder.py` , or write it here.

Your product function _____

1. two inputs: (a, b)
2. return a times b

product(6, 7) expected: 42

What did YOU get? _____ ☐ it matches

5 Rung 5: the four functions from the slide. These four are your target. The names must be exactly `add` , `subtract` , `multiply` , `divide` .

```
def add(a, b):  
    return _____ # +?  
  
def subtract(a, b):  
    return _____ # -?  
  
def multiply(a, b):  
    return _____ # *?  
  
def divide(a, b):  
    return _____ # /?
```

Your four return lines _____

// +? -? *? /? write each return line here, then code it in `ladder.py`

Run the file once, then check all four in the console:

add(2, 3) expected: 5

What did YOU get? _____ ☐ it matches

subtract(10, 4) expected: 6

What did YOU get? _____ ☐ it matches

multiply(6, 7) expected: 42

What did YOU get? _____ ☐ it matches

divide(9, 2) expected: 4.5

What did YOU get? _____ ☐ it matches

Your tracker. Tick only when the console value matches the chip.

Function	test worked?	notes
add		
subtract		
multiply		
divide		

Gate: all four must run in one file, `ladder.py`, before the class builds `calculator.py`. It calls them by these exact names. One wrong name and it finds nothing.

Part 3 Use them (rungs 6–7)

6 Rung 6: calls inside calls. Guess first, then run each line in the console.

Inside-out rule: do the bracket nearest the middle first, then use its answer in the outer call.

code	what I think happens	what actually happened	✓
<code>multiply(2, 5)</code>	Example: the console shows 10.	Example: it showed 10.	✓
<code>add(multiply(3, 4), 2)</code>			
<code>divide(add(6, 8), 2)</code>			

7 Rung 7: the stretch

Stretch: `power(a, b)` gives `a` to the power `b`. Put it in `ladder.py` and run it.

// hint: `a ** b` means multiply `a` by itself `b` times

`power(2, 10)` expected: 1024

What did YOU get? _____ ☐ it matches

Also try `divide(5, 0)` in the console. Write down **exactly** what appears. You do not need to fix it; reading errors is part of the job.
