

Programming Basics

- 1. Without running the code, determine the value and type of each expression:

Expression	Value	Type
17 // 5	3	int
23 % 6	5	int
9 / 4	2.25	float
8 ** 2	64	int
"ha" * 7 + "!"	"hahahahahahaha!"	str
int("12") + 30	42	int
str(15) + "0"	"150"	str

- 2. Trace the following code and write what it prints.

Python

```
x = 23
y = 6
x = x % y
y = y + x
z = y / x
print(x, y, z)
```

5 11 2.2

- 3. Trace the following code and write what it prints.

Python

```
n = 7
total = 0.5
n = n * 2 - 3
total = total + n / 4
print(n, total)
```

11 3.25

- 4. Trace the following code and write what it prints.

Python

```
p = 18
q = 4
r = p // q
s = p % q
t = r > s
print(r + s, t)
```

6 True

- 5. For each print statement, what are the resulting values and types of each expression? Enter your answers below.

Python

```
a = 7
b = 2
c = 3.0

print(a / b) #1
print(a / c) #2
print(a > b) #3
print(str(a) + str(b) + str(c)) #4
print(a + b + c) #5
```

#	Value	Type
1	3.5	float
2	2.3333333333333335	float
3	True	bool
4	"723.0"	str
5	12.0	float

- 6. For each of the following, write what it prints, then write whether it ends with an error. If it does, name the error type and the line where it happens.

Python

```
# A
a = 9
b = a - 9
print(a + 1)
print(a // b)
print(b)

# B
```

```
print(4 * 3)
print(count + 1)
count = 2
```

```
# C
x = 5
y = 2
print(x % y)
print(x y)
```

```
# D
k = 3
print(k > 1)
print(k ** 2)
print(k // 0.5)
```

```
# E
print(2 + 3
print(9)
```

```
# F
v = 4
w = v + 1.5
print(w)
print(w + z)
```

	What does it print?	Does it end with an error? If so, what type and what line?
A	10	Yes — ZeroDivisionError (runtime error) on <code>print(a // b)</code> . <code>b</code> is <code>9-9 = 0</code> . <code>print(b)</code> never runs.
B	12	Yes — NameError (runtime error) on <code>print(count + 1)</code> . <code>count</code> is not assigned until the next line.
C	Nothing at all	Yes — SyntaxError on <code>print(x y)</code> . Syntax errors are caught before any code runs, so <code>print(x % y)</code> never executes.

D	True 9 6.0	No error. Note <code>3 // 0.5</code> gives 6.0 (a float), not 6.
E	Nothing at all	Yes — <code>SyntaxError</code> on <code>print(2 + 3</code> (unclosed parenthesis). Nothing runs.
F	5.5	Yes — <code>NameError</code> (runtime error) on <code>print(w + z)</code> . <code>z</code> was never defined.

Function Calls

- 1. Fill out the table below

Function Call	Argument	Returned Value
<code>math.ceil(1.7)</code>	1.7	2
<code>random.randint(1,6)</code>	1 and 6	a random int from 1 to 6 (inclusive) [VARIES EACH RUN]
<code>pow(2,3)</code>	2 and 3	8
<code>print("15110")</code>	"15110"	None — print returns nothing; displaying 15110 is a side effect

- 2. In the table above, there are two function calls that require external libraries. Please (1) point out which two functions they are, and (2) write the appropriate import statement for each of them.

Function Call	Import Statement
<code>math.ceil(1.7)</code>	<code>import math</code>
<code>random.randint(1,6)</code>	<code>import random</code>

- 3. You have written out the returned value of the function calls above; how would you check if the returned value is correct? Choose one function call above and write the `assert` functions to check if its returned value matches your answer.
(Multiple correct answers)

Assert Command	Function call with the test input	op	Expected output given the input
<code>assert</code>	<code>math.ceil(0.1)</code>	<code>==</code>	1
<code>assert</code>	<code>math.ceil(1.7)</code>	<code>==</code>	2
<code>assert</code>	<code>math.ceil(7)</code>	<code>==</code>	7

- 4. Please answer:
 - (1) How many arguments are in the function call `msg_to_santa(name, age, wish)`?
 - (2) What keyword denotes the start of a function?
 - (3) What happens when calling a function with no return statement?
 - (4) What's the difference between `print` and `return` statements inside a function?

1. 3 — name, age, and wish
2. `def`
3. It still runs, but the call evaluates to `None`.
4. `print` displays text to the user (a side effect) and does not hand a value back; `return` sends a value back to the caller and immediately ends the function. A printed value cannot be stored in a variable or used in further computation; a returned value can.

- 5. What does `print` mean? What does the statement `print(3)` do? What does it return?

`print` is a built-in function that displays a value in the output/console. `print(3)` displays the character 3 to the user. It returns `None` — the display is a side effect, not a returned value.

- 6. What is the output from `print(1/3)`? Why do we get this output?

Answer: 0.3333333333333333

The `/` operator always produces a float. Computers store floats in a finite number of binary bits, so `1/3` (which repeats forever) cannot be stored exactly. Python displays the closest representable value, about 16 significant digits.

- 7. What do each of these functions evaluate to? What does it return (write None if nothing)? Does it have any side effects (write No if not)?

Function call:	Returns:	Side effects:
<code>abs(-2)</code>	2	No
<code>pow(2, 3)</code>	8	No
<code>round(12.4567, 2)</code>	12.46	No
<code>int("4")</code>	4	No
<code>float(3)</code>	3.0	No
<code>str(98.9)</code>	"98.9"	No
<code>bool(0)</code>	False	No
<code>type(4 + 3.0)</code>	<class 'float'>	No

- 8. What will happen when you execute the following code?

```
num = input("Enter a number to be doubled:")
doubled_num = num * 2
print(doubled_num)
```

It does not double the number. `input()` always returns a string, so `* 2` performs string repetition, not multiplication. If the user types "5", it prints "55"; if they type "12", it prints "1212". No error occurs.

- 9. What is going to be printed once each of the following code blocks is executed?

#code block 1

```
a = -4
b = 2
c = abs(a)
x = a * c
print(x)
```

#code block 2

```
a = 3
b = 2
```

```
x = pow(a, b)
print(x)
```

```
x = pow(b, a)
print(x)
```

Block 1: -16 (c = abs(-4) = 4, so x = -4 * 4 = -16)
Block 2: 9 then 8 (pow(3,2) = 9; pow(2,3) = 8)

- 10. What will the following evaluate to?

```
round(pow(abs(-35), 1/2), 2)
pow(round(34/4,0), abs(3))
```

round(pow(abs(-35), 1/2), 2) = 5.92
pow(round(34/4,0), abs(3)) = 512.0

- 11. Trace through the following code. What are the values of a and b at the end, once all the code has executed? What is going to be printed at the end, once all the code has executed?

```
a = 4
b = 15

a = b // a
a = b / a
b = b % 4
print(a + b)
```

```
a = 5.0
b = 3

prints: 8.0
```

- 12. Consider this function definition

```
def mystery(num):
    return abs(num) + num
```

Imagine we ran `mystery(-5)`.

- Does this call have any argument(s)? If yes, list it/them.
- Does this call have any returned value(s)? If yes, list it/them.
- Does this call have any side effect(s)? If yes, list it/them.

```
One argument: -5
Returned value: 0
No side effects: nothing else is printed or altered
```

- 13. Suppose we execute `y = mystery(x)` where `type(x) == int`. Which expressions evaluate to True?
 - `y > 0`
 - `not (y < 0)`
 - `x <= y`
 - `y == 0`
 - `type(y) == int`
 - `type(y) == float`

#	Expression	True?	Why
1	<code>y > 0</code>	No	If $x \leq 0$ then $y = 0$, which is not > 0
2	<code>not (y < 0)</code>	Yes	y is $\text{abs}(x)+x$, which is $2x$ when $x \geq 0$ and 0 when $x < 0$ — never negative
3	<code>x <= y</code>	Yes	If $x < 0$, $y = 0 > x$. If $x \geq 0$, $y = 2x \geq x$
4	<code>y == 0</code>	No	Only true when $x \leq 0$
5	<code>type(y) == int</code>	Yes	<code>abs()</code> of an int is an int, and $\text{int} + \text{int}$ is an int
6	<code>type(y) == float</code>	No	No division or float is involved

- 14. Using variables `x` and `y` provided below, compute their total, the absolute magnitude of their difference, `x` exponentiated to `y`, and the rounded quotient of `x` divided by `y`:

`x = 5`

`y = 2`

Complete your implementation here

`total_val = x + y` # 7

`abs_diff = abs(x - y)` # 3

`power_val = pow(x, y) OR x ** y` # 25

`rounded_div = round(x / y) OR round(x/y, 0)` # 2

- 15. Using the input function, obtain a radius value from the user to calculate the corresponding volume of a sphere (where $V = \frac{4}{3} * \pi * r^3$) (assume the math library was already imported):

```
radius = float(input("Enter a radius: "))
```

```
volume = (4/3) * math.pi * (radius ** 3)
```

- 16. Construct a Python statement to generate a **random integer** from 1 through 10, calculate its **square**, and output the evaluated result:

```
import random
```

```
print(pow(random.randint(1,10), 2))
```

Function Definitions

- 1. Once the code below executes, what would be printed?

Python

```
x = 10
def test():
    x = 5
    print("Inside:", x)
test()
print("Outside:", x)
```

Inside: 5
Outside: 10

- 2. Given the following functions and function call, what would the variable result hold when printed in the end?

Python

```
def addOne(x):
    return x + 1
def square(x):
    return x * x

result = square(addOne(3))
print(result)
```

16

- 3. What is printed once the code below executes?

Python

```
def sayHi():  
    print("Hi!")  
  
x = sayHi()  
print(x)
```

Hi!
None

- 4. Would the following code result in an error? If yes, determine the type of error, and if not, what would be printed?

Python

```
x = 5  
  
def foo():  
    x = 10  
    return x  
  
print(foo())  
print(x)
```

10
5
(No error)

- 5. What will the function calls print in the end?

Python

```
def f(x):  
    return x + 2  
  
def g(x):  
    return f(x) * 3  
  
def h(x):  
    return g(x) - 1  
  
print(h(2))
```

11

- 6. What is the difference between defining a function and calling a function? Feel free to give an example.

Defining a function (with def) creates it and stores the instructions under a name. Calling a function (with its name and parentheses, e.g. square(4)) actually runs those instructions on specific argument values and produces a returned value. A function can be defined once and called many times; a function that is defined but never called never runs.

- 7. Write a function by rearranging the lines of code below that takes text input from the user and displays the input in lower case.

Correct line #	Mixed code lines
3	<code>print(userInput.lower())</code>
1	<code>def printInputLower():</code>
2	<code>userInput = input("Please enter some text input:")</code>

- 8. Write the function `diff21(n)` which, given an integer `n`, returns the absolute difference between `n` and 21. Call the function with `n = 10` and print the returned value.

Correct line #	Mixed code lines
2	<code>abs_diff = abs(n - 21)</code>
1	<code>def diff21(n):</code>
4	<code>print(diff21(10))</code>
3	<code>return abs_diff</code>

Prints: 11

- 9. What will the following code print? Explain in a few words what is happening

```
x = "Hello World"
def fun1():
    x = 2
    print("inside the function x has the value:", x)

fun1()
print("outside the function x has the value:", x)
```

inside the function x has the value: 2
outside the function x has the value: Hello World

(The x inside fun1 is a local variable that shadows the global x only within the function
Reassigning it does not change the global)

- 10. Provided the following Python code, please answer the following questions.
 - What is the scope of the variable name?
 - What is the scope of the variable date?

```
1. date = "January 28th"
2. def welcome(name):
3.     print("Welcome, " + name + "!")
4.     print("Today is" + date + ".")
5. print("Goodbye, " + name + ".")
```

name: local to welcome
date: global variable

- 11. Provided the following Python code, trace the function calls. What will be printed after line 6 is executed?

```
1. def foo(x,y,z):
2.     return x*y+z

3. def bar(x):
4.     a = foo(x, x+2, x*2)
5.     return a**2

6. print(bar(3))
```

441

- 12. For the following code, answer the questions below:
 - What will this block of code print once it stops running?
 - List all the function calls that occur in this code block with their name, argument value(s), and returned value. If there is no name / argument / returned value, leave the space blank. Do not include any calls to built-in functions (like print) in the table.
 - List all the global and local variables in the code, and determine to which function the local variables belong.

```
def fun1(x):
    tmp = x * 2
    x = 3
    print("fun1:", x)
    x = "WOW"
    return x

def fun2(x):
    x = x * 3
    y = int(x) + 2
    print("fun2:", fun1(x))
    return y

x = "5"
result = fun2(x)
print("End: ", x, result)
```

```
fun1: 3
fun2: WOW
End: 5 557
```

Name	Argument value(s)	Returned value
fun2	"5"	557
fun1	"555"	"WOW"

Scope	Variables
Global	x, result
Local to fun1	x (parameter), tmp
Local to fun2	x (parameter), y

- 13. What will be printed when we run the following code?

```
def test(x):  
    print("A:", x)  
    return x + 5
```

```
y = 2  
print("B:", y)  
z = test(y + 1)
```

```
B: 2  
A: 3
```

- 14. What will the following code print once it is executed? Explain in a few sentences why.

```
1. def outer(x):  
2.     y = x + 2  
3.     print("outer y:", y)  
4.     return inner(y) * 2  
5.  
6. def inner(x):  
7.     y = x * 2  
8.     print("inner y:", y)  
9.     return y  
10.  
11. print(outer(4))
```

```
outer y: 6  
inner y: 12  
24
```

- 15. Write a function `dice(n)` which simulates a dice roll for a dice with `n` sides and prints the result. Hint: Using the `random` library is helpful here! Call the function with 3 different values of `n` and see the results!

Correct line #	Mixed code lines
3	<code>roll = random.randint(1, n)</code>
1	<code>import random</code>
2	<code>def dice(n):</code>
4	<code>return roll</code>

- 16. Write a function `rectangleFun(w, h)` which takes in the width and height of a rectangle, prints the perimeter, and returns the area.

Correct line #	Mixed code lines
4	<code>return area</code>
2	<code>print(w*2 + h*2)</code>
3	<code>area = w * h</code>
1	<code>def rectangleFun(w, h):</code>

Conditionals and Errors

- 1. What would each of the following expressions evaluate to?

Python

True and not False #1

not (True and not False) #2

(False or True) and True #3

not True and (False or True) #4

#	Expression	Value	Why
1	True and not False	True	not False = True; True and True = True
2	not (True and not False)	False	inner is True (from #1); not True = False
3	(False or True) and True	True	(False or True) = True; True and True = True
4	not True and (False or True)	False	not binds tighter than and, so this is (not True) and True = False and True = False

- 2. If $x = 6$ and $y = 7$, what will the following expressions evaluate to

Python

```
(x < y) and (x > 3) #1
(x == 9) or (y == 7) #2
not (x == 6 and y > 5) #3
(x != 6) or (y >= 7) #4
(x < 10) and not (y < 7) #5
```

#	Expression	Value	Why
1	$(x < y) \text{ and } (x > 3)$	True	$6 < 7$ True; $6 > 3$ True
2	$(x == 9) \text{ or } (y == 7)$	True	first False, second True; or needs only one
3	$\text{not } (x == 6 \text{ and } y > 5)$	False	inner: True and True = True; not True = False
4	$(x != 6) \text{ or } (y >= 7)$	True	first False, second True ($7 \geq 7$)
5	$(x < 10) \text{ and not } (y < 7)$	True	$6 < 10$ True; $(7 < 7)$ is False, not False = True

- 3. Consider the following code:

Python

```
def yippee(a,b,c):
    if a % 2 == 1:
        if b < 20:
            print("A")
        elif b < 60:
            print("B")
        if c > 3:
            print("C")
    elif b < 15:
        if a >= 3 and b < 20:
            print("D")
```

```

        if a < 20 or b > 4:
            print("E")
    else:
        if c == 5:
            print("F")
        else:
            print("G")
    print("H")

```

For each of the function calls below, write in the cell under it what will be printed once the code executes.

yippee(8, 20, 5)	yippee(7, 25, 6)	yippee(4, 8, 3)
F H	B C	D E

Is there an input that would cause the code to print both “A” and “B”? If yes, give an example. If not, explain why.

No. A and B are in an if / elif pair on the same variable b, and an elif branch is only tested when the preceding if was False. Since b cannot be both < 20 and not < 20 at the same time, exactly one of the two can ever run.

- 4. For each example below, identify the error type, error name, and why it happens:

Code	Error Type	Error Name	Issue
<pre>Python x = 5 if x > 3 print("Yay")</pre>	Syntax error	SyntaxError	Missing colon (:) at the end of the if condition. Nothing runs at all.
<pre>Python x = 0 print(100 / x)</pre>	Runtime error	ZeroDivisionError	Division by zero is undefined. The code parses fine but crashes when it executes.
<pre>Python #this function should return x times y def multiply(x, y): return x ** y</pre>	Logical error	(none — no error is raised)	** is exponentiation, not multiplication. The code runs and returns a value, but the wrong one: multiply(3,4) gives 81 instead of 12.
<pre>Python a = "Hello" b = "World" c = 5 print(a + b + c)</pre>	Runtime error	TypeError	Cannot concatenate a str and an int. "HelloWorld" + 5 has no meaning; c would need str(c).
<pre>Python import rand</pre>	Syntax error	SyntaxError	The call to print is missing its closing parenthesis. Nothing runs.

<pre>print(random.randint (1, 5)</pre>			
--	--	--	--

- 5. What type of error does this block of code produce, if any?

Python

```
x = 5  
if x > 10:  
    print("Big")  
else:  
    print("Big")
```

No error is raised!

- 6. What type of error does this block of code produce, if any?

Python

```
x = 5  
if x > 3  
    print("hi")
```

SyntaxError, missing the colon after the condition.

7. What type of error does this block of code produce, if any?

Python

```
x = 2  
y = 2  
print("2 + 2 =", x + y + 1)
```

No error is raised. This could be considered a logical error (2 + 2 does not equal 5), but python doesn't catch that error without assert statements.

- 8. Fill in the blank the code below:

In basketball, three plays score points: a three-point shot, a two-point shot, and a one-point free throw. You just watched a basketball game between the Apples and Bananas, and recorded the number of successful three-point, two-point, and one point plays for each team. If the game was won by the Apples we want to print A, if the game was won by the Bananas we want to print B, and if the game was a tie we want to print T.

```
apple_three = int(input("Enter the # of three points for apple:"))
apple_two = int(input("Enter the # of two points for apple:"))
apple_one = int(input("Enter the # of one points for apple:"))

banana_three = int(input("Enter the # of three points for banana:"))
banana_two = int(input("Enter the # of two points for banana:"))
banana_one = int(input("Enter the # of one points for banana:"))

apple_total = apple_three * 3 + apple_two * 2 + apple_one
banana_total = banana_three * 3 + banana_two * 2 + banana_one

if apple_total > banana_total:
    print("A")
elif banana_total > apple_total:
    print("B")
else:
    print("T")
```

- 9. Fill in the blank the code below:

Write the function `sleep_in(weekday, vacation)`: where the parameter `weekday` is `True` if it is a weekday, and the parameter `vacation` is `True` if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return `True` if we sleep in. For example:

```
sleep_in(False, False) → True
sleep_in(True, False) → False
sleep_in(False, True) → True
```

```
def sleep_in(weekday, vacation):
    if not weekday or vacation:
        return True
    return False
```

- 10. Arrange the following lines of code.

Write the function `sum_double(a, b)`: Given two int values, return their sum. Unless the two values are the same, then return double their sum. For example:

```
sum_double(1, 2) → 3
sum_double(3, 2) → 5
sum_double(2, 2) → 8
```

Correct line #	Mixed code lines
3 or 2	if a == b:
2 or 5	sum = a + b
5 or 4	return double_sum
1	def sum_double(a, b):
4 or 3	double_sum = (a + b) * 2
6	return sum

Loops

1. For the code below, what are the start value, stop value, and update step of the loop iterator `i`, respectively?

Python

```
for i in range(15, 2):  
    print(i)
```

Value	Answer
Start	15
Stop	2
Update	+1 (the default, since no third argument is given)

2. Trace the following code chunk

Python

```
L = [1, 5, 110]  
for i in range(len(L)):  
    if i % 2 == 0:  
        L[i] = i+1  
        print(L[i])
```

1
3

(Not covered): L ends up as [1, 5, 3]

Strings and Lists

1. What does each line evaluate to?

Python

```
print("Hi" + "There")
print([1, 2] + [3])
print("Go" * 3)
print(["Go"] * 3)
```

Line	Output
print("Hi" + "There")	HiThere
print([1, 2] + [3])	[1, 2, 3]
print("Go" * 3)	GoGoGo
print(["Go"] * 3)	['Go', 'Go', 'Go']

2. Print the outputs of each line:

Python

```
s = "abcdef"
lst = [10, 20, 30, 40, 50]

print(s[1:4])
print(lst[2:])
print(s[:3])
print(lst[:len(lst)-1])
print(s[::2])
print(lst[1::2])
```

Line	Output	Why
print(s[1:4])	bcd	indices 1, 2, 3 — the end index is excluded
print(lst[2:])	[30, 40, 50]	from index 2 to the end
print(s[:3])	abc	from the start through index 2
print(lst[:len(lst)-1])	[10, 20, 30, 40]	everything except the last item (len is 5, so stop at 4)
print(s[::2])	ace	whole string, step 2 — indices 0, 2, 4
print(lst[1::2])	[20, 40]	start at 1, step 2 — indices 1, 3

3. Fill in the blank and then determine roughly what this function will do:

```
Python
s = "hello"
result = ""
for i in range(len(s)):
    if s[i] != "l":
        result = result + s[i]
print(result)
```

This function builds a new string containing every character of s except the letter "l"

4. What will each line print?

```
Python
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

print(matrix[0])
print(matrix[1][2])
print(matrix[2][0])
```

Line	Output	Why
print(matrix[0])	[1, 2, 3]	one index gives a whole row (which is itself a list)
print(matrix[1][2])	6	row 1 is [4,5,6]; its index 2 is 6
print(matrix[2][0])	7	row 2 is [7,8,9]; its index 0 is 7

5. Complete this function by filling in the blank:

Python

```
def sumMatrix(matrix):  
    total = 0  
    for row in range(len(matrix)):  
        for col in range(len(matrix[row])):  
            total = total + matrix[row][col]  
    return total
```

6. Given this string, how would you get the following?

Python

```
s = "abcdef"
```

```
# abc
```

```
# bcd
```

```
# def
```

```
# ace
```

Target	Answer(s)
"abc"	s[0:3] or s[:3]
"bcd"	s[1:4] — only one natural form
"def"	s[3:6] or s[3:] or s[len(s)-3:]
"ace"	s[0:6:2] or s[::2] or s[0::2]

7. Given this list, how would you get each subset?

Python

```
lst = [5, 10, 15, 20, 25, 30]
```

```
# [15, 20, 25]
```

```
# [10, 20, 30]
```

Target	Answer(s)
[15, 20, 25]	<code>lst[2:5]</code>
[10, 20, 30]	<code>lst[1::2]</code> or <code>lst[1:6:2]</code> or <code>lst[1:len(lst):2]</code>

8. What would happen if you ran this line of code? Briefly explain.

Python

```
print(s[len(s)])
```

It crashes with an `IndexError` ("string index out of range")

9. Below are the lines of code that make up a complete function called `squaresOfSmall`. The function takes a list of integers, `nums`. It builds a new list containing the square of every value less than 10. Finally, it prints the count and returns the list.

Arrange the lines of code in the correct order by putting the correct line number next to each.

Correct line #	Mixed code lines
8	<code>big = big + 1</code>
10	<code>return squares</code>
4	<code>for n in nums:</code>
1	<code>def squaresOfSmall(nums):</code>
5	<code>if n < 10:</code>
9	<code>print("Ignored", big, "large values.")</code>
7	<code>else:</code>
6	<code>squares = squares + [n * n]</code>
2 or 3	<code>big = 0</code>

3 or 2	squares = []
--------	--------------

10. For the following block of code, what are the start, stop, and update values for the loop control variable?

```
Python
i = 10
while i > 0:
    print(i)
    i -= 1
```

Start	10
Stop	0 — the loop keeps going while $i > 0$, so it stops once i reaches 0 (the last value printed is 1)
Update	subtract 1 each iteration

What will be printed when the above code runs?

```
10
9
8
7
6
5
4
3
2
1
```

11. Given the following block of code, what is printed when we run this code?

Python

```
def findDivisible(L, k):
    result = []
    for i in range(len(L)):
        currVal = L[i]
        if currVal % k == 0:
            result = result + [currVal]
    return result

def printPairs(L):
    k = L[0]
    L1 = findDivisible(L, k)
    for i in range(k):
        for j in range(len(L1)):
            print(i, L1[j])
    return L1

print("Out:", printPairs([2,7,8,6]))
```

0 2

0 8

0 6

1 2

1 8

1 6

Out: [2, 8, 6]

Testing

1. Assume you have written a function that works correctly when you run it once. Why is that not enough to conclude your code works correctly?

One successful run only shows the function works for that one input. Bugs very often appear only for other inputs: negative numbers, zero, empty structures, values on either side of a decision point in a conditional, or larger inputs. A single test also cannot distinguish code that is right from code that happens to give the right answer for that case by coincidence. Good testing needs multiple tests covering varied inputs, varied outputs, and edge cases.

2. If you currently have a program that works correctly, but then modify it to add a new feature, must you still go through with testing the code? Why or why not?

Yes. You must test both the new feature AND the parts that already worked. Changes can break existing behaviour: a shared helper function, a changed variable, or a new branch can affect old functionality. Re-running the previous tests after a change is how you catch that.

3. Why is testing important?

- A. It proves that the program has no bugs whatsoever.
- B. It checks whether code meets its expectations and works correctly.
- C. It makes programs run faster.
- D. It makes sure you don't need to understand the code when using it.

4. What is the difference between unit testing and system testing?

Unit testing checks one small piece of the codebase, usually a single function, in isolation, typically with a set of assert statements covering varied and edge-case inputs. System testing runs the entire program from beginning to end on an input whose expected output you already know, to check the system as a whole meets its goal. Unit tests pinpoint which function is broken; system tests catch problems that only appear when the parts work together.

5. You've written the following function (that, unbeknownst to you, is buggy!). It should return the sum of all even integers from 1 to n, inclusive of n. Write a series of unit tests that identifies what kinds of inputs make the function work as expected and which kinds of inputs it fails on.

Python

```
def sum_of_evens(n):  
    total = 0  
    for i in range(2, n, 2):  
        total += i  
    return total
```

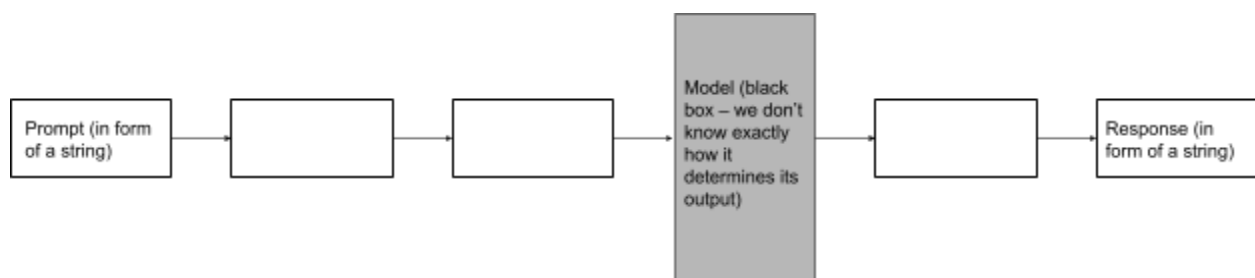
assert command	function call	op	expected output
assert	sum_of_evens(5)	==	6 — PASSES (2+4)
assert	sum_of_evens(7)	==	12 — PASSES (2+4+6)
assert	sum_of_evens(1)	==	0 — PASSES (no evens; edge case)
assert	sum_of_evens(6)	==	12 — FAILS, returns 6
assert	sum_of_evens(2)	==	2 — FAILS, returns 0 (smallest even edge case)

Demystifying Computers

1. Match the following generative AI vocabulary with their definitions.

Term	Correct definition
Token	small chunk of text, often a word, used by GenAI to understand your prompt
Probabilistic output tokens	a series of tokens and their associated probabilities, taking into account the already chosen tokens
Response	the final response presented to the user, generated by the probabilistic model
Prompt	input to give a GenAI system (a question, instruction, or description of what you want)
Token embeddings	words turned into numbers to capture their meaning (strings → numbers)
Model	the underlying system that a GenAI uses to generate responses; trained on massive amounts of data to predict the most probable next token

2. Using the framework given below, fill out the name of each of the steps that a generative artificial intelligence model takes to process and respond to a given prompt.



Prompt → Tokens → Token embeddings → Model → Probabilistic output tokens → Response

3. You were strolling along the beach when you found a message in a bottle. Excitedly you open it up to find the following three bits of binary.

00101101 10111001 01000110

Give three options for what the binary could represent.

Binary	Decimal
00101101	45
10111001	185
01000110	70

- Three whole numbers: 45, 185, and 70.
- A single colour, as an RGB value: red=45, green=185, blue=70 — a medium green.
- Three text characters under some character encoding. Careful: 45 is "-" and 70 is "F" in ASCII, but 185 is outside standard 7-bit ASCII (0–127), so this only works under an 8-bit encoding such as Latin-1.
- Three greyscale pixel values (how dark each of three pixels is, 0–255).
- Many more acceptable answers.

Circuits and Gates

1. Convert the following boolean expressions to truth tables and circuits

- not A

A	not A
0	1
1	0

- A xor B

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

- A and B

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

- A or B

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

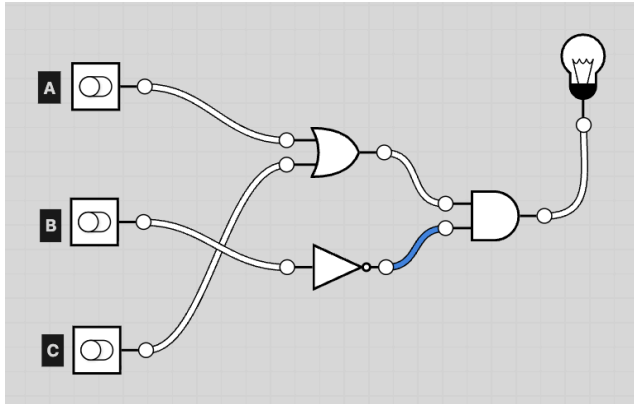
- (not A) and (B or (not C))

A	B	C	(not A) and (B or (not C))
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

- (A and (not B)) or (B and C)

A	B	C	(A and (not B)) or (B and C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

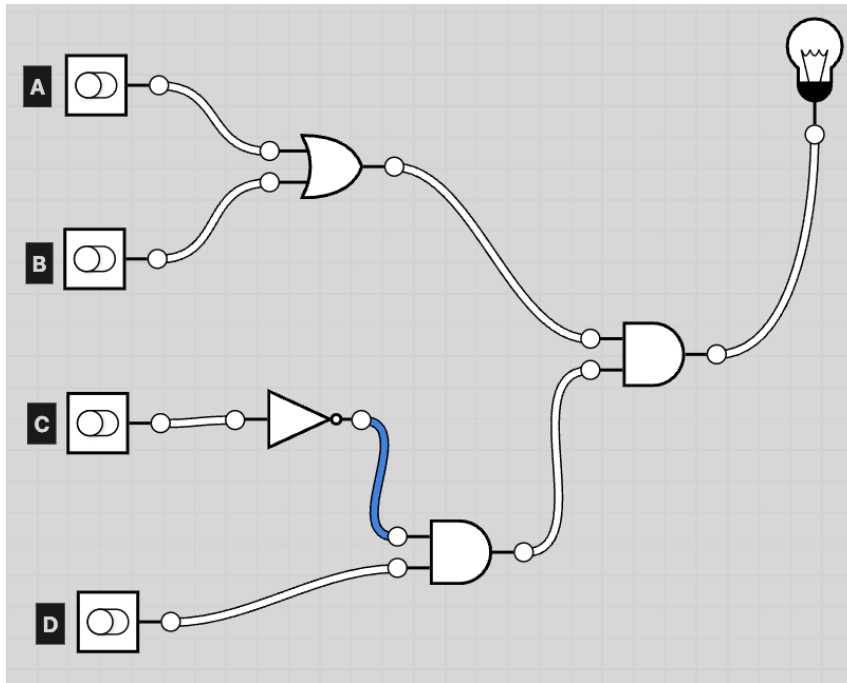
2. Write the boolean expression and truth table for the circuit below.



(A xor C) and (not B)

A	B	C	(A xor C) and (not B)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

3. Write the boolean expression and truth table for the circuit below.



(A or B) and (not C and D)

A	B	C	D	(A or B) and (not C and D)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0

1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

4. Consider two one-bit inputs, X and Y. Complete the circuit below based on all the possible combinations of X and Y a circuit can take in.

X	Y	X + Y	Carry	Sum
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	10	1	0

5. A 4-bit adder is built by connecting four full adders. Knowing this, calculate $11 + 3$ and record the steps below.

Bit Position	X	Y	C _{in}	Sum	C _{out}
2 ⁰	1	1	0	0	1
2 ¹	1	1	1	1	1
2 ²	0	0	1	1	0
2 ³	1	0	0	1	0

2^0 : $1+1+0 = 2 \rightarrow$ Sum 0, carry 1.
 2^1 : $1+1+1 = 3 \rightarrow$ Sum 1, carry 1.
 2^2 : $0+0+1 = 1 \rightarrow$ Sum 1, carry 0.
 2^3 : $1+0+0 = 1 \rightarrow$ Sum 1, carry 0.

$11 + 3 = 14$ (binary 1110)

6. Put the following in order from most concrete to most abstract representation:

- Full adder
- Individual AND/OR/XOR gates
- 4-bit adder
- Binary addition algorithm

Order	Item
1	Individual AND/OR/XOR gates
2	Full adder
3	4-bit adder
4	Binary addition algorithm

Ethics of Algorithm Design

1. Sometimes algorithms are used to make decisions that will directly affect the life of an individual. Some examples of where these algorithms are used include; facial recognition, criminal risk assessment, college admissions, and kidney matching. Pick one of these examples, or pick your own, and describe some complicating factors the algorithm would have to consider. Then, compare pros and cons for having decisions related to these topics made by humans, or by the algorithms.

Sample answer (kidney matching):

Complicating factors the algorithm would have to weigh: medical compatibility (blood and tissue type), how long a candidate has been waiting, how urgent their need is, expected years of life gained, geographic distance and organ transport time, whether the candidate has previously donated a kidney, and the likelihood the transplant succeeds. Each of these embeds a value judgement. For example, weighting expected life-years favours younger patients, and weighting prior donation is a fairness rule rather than a medical one. Someone has to decide the relative weights, and that choice is not itself a medical fact.

Pros of algorithmic decisions: Algorithmic decisions are consistent: the same case gets the same answer regardless of who reviews it or what time of day it is; scalable to thousands of candidates; makes the criteria explicit and therefore debatable and auditable; avoids individual human prejudice on any single case.

Cons of algorithmic decisions: the algorithm encodes whoever wrote it, so bias moves rather than disappearing; historical training data can carry forward past discrimination (the ProPublica analysis of the COMPAS risk-assessment algorithm found exactly this); many systems are proprietary and secret, so those affected cannot see or challenge the reasoning; and an algorithm cannot weigh unusual context a human would notice.

—

New surveillance technologies have sparked scrutiny of algorithmic ethics and use. Notably, policing cameras produced by [Flock Safety](#) have been installed in many US cities.

Police departments, a recent CrimRxiv study, and Flock itself claim that such technologies improve policing. They cite that:

- Flock cameras are involved in 1 in 5 police cleared cases
- Vehicle tracking reportedly helps locate about 10,000 missing persons every year

- About 42% of Flock customers report an estimated 50% of stolen vehicles in their jurisdictions are recovered with the help of Flock cameras
- Deploying automated license plate reader (ALPR) networks reduced motor vehicle theft by 11% .

Sources:

<https://www.flocksafety.com/blog/how-effective-is-flock>

<https://www.crimrxiv.com/pub/zleg04q3/release/1>

Even if effective, critics contend the algorithms used by Flock cameras are biased and/or unethical. Common sources of disapproval include:

- Where ALPR cameras are installed, which is a part of the algorithm's execution and training. If you're interested, look at the graphics in [this study](#) of Virginia municipalities.
- The accuracy of ShotSpotter and license plate reading technology for reporting gunshots and vehicle thefts.
- Abuses of flock databases by law enforcement to discriminate against certain groups or individuals.

Sources:

<https://www.eff.org/deeplinks/2025/12/effs-investigations-expose-flock-safety-surveillance-abuses-2025-review>

<https://www.gadgetreview.com/one-in-three-alerts-wrong-high-error-rates-and-data-concerns-force-cities-to-drop-flock>

<https://www.usatoday.com/story/news/crime/2026/07/11/flock-gunshot-detector-privacy-concerns/90757260007/>

Considering the information above and your own knowledge:

2. What place does new algorithm-driven surveillance technology have in our society? If Flock cameras are here to stay, how might we address algorithmic concerns?