```python
# Says hello to someone

s = input()
print(f"hello, {s}")
```

```python
# A program


def main():
    print("hello, world")


if __name__ == "__main__":
    main()
```

```python
class Dictionary:
    """Implements a dictionary's functionality"""

    def __init__(self):
        self.words = set()

    def check(self, word):
        """Return true if word is in dictionary else false"""
        return word.lower() in self.words

    def load(self, dictionary):
        """Load dictionary into memory, returning true if successful else false"""
        file = open(dictionary, "r")
        for line in file:
            self.words.add(line.rstrip("\n"))
        file.close()
        return True

    def size(self):
        """Returns number of words in dictionary if loaded else 0 if not yet loaded"""
        return len(self.words)

    def unload(self):
        """Unloads dictionary from memory, returning true if successful else false"""
        return True
```

```python
#!/usr/bin/env python

import re
import sys
import time

from dictionary import Dictionary

# Maximum length for a word
# (e.g., pneumonoultramicroscopicsilicovolcanoconiosis)
LENGTH = 45

# Default dictionary
WORDS = "dictionaries/large"

# Check for correct number of args
if len(sys.argv) != 2 and len(sys.argv) != 3:
    print("Usage: speller [dictionary] text")
    sys.exit(1)

# Benchmarks
time_load, time_check, time_size, time_unload = 0.0, 0.0, 0.0, 0.0

# Determine dictionary to use
dictionary = sys.argv[1] if len(sys.argv) == 3 else WORDS

# Load dictionary
d = Dictionary()
before = time.process_time()
loaded = d.load(dictionary)
after = time.process_time()

# Exit if dictionary not loaded
if not loaded:
    print("Could not load $dictionary.")
    sys.exit(1)

# Calculate time to load dictionary
time_load = after - before

# Try to open text
text = sys.argv[2] if len(sys.argv) == 3 else sys.argv[1]
file = open(text, "r", encoding="latin_1")
if not file:
    print("Could not open {}.".format(text))
```

```
        d.unload()
        sys.exit(1)

# Prepare to report misspellings
print("\nMISSPELLED WORDS\n")

# Prepare to spell-check
word = ""
index, misspellings, words = 0, 0, 0

# Spell-check each word in file
while True:
    c = file.read(1)
    if not c:
        break

    # Allow alphabetical characters and apostrophes (for possessives)
    if re.match(r"[A-Za-z]", c) or (c == "'" and index > 0):

        # Append character to word
        word += c
        index += 1

        # Ignore alphabetical strings too long to be words
        if index > LENGTH:

            # Consume remainder of alphabetical string
            while True:
                c = file.read(1)
                if not c or not re.match(r"[A-Za-z]", c):
                    break

            # Prepare for new word
            index, word = 0, ""

    # Ignore words with numbers (like MS Word can)
    elif c.isdigit():

        # Consume remainder of alphanumeric string
        while True:
            c = file.read(1)
            if not c or (not c.isalpha() and not c.isdigit()):
                break

        # Prepare for new word
        index, word = 0, ""

    # We must have found a whole word
    elif index > 0:

        # Update counter
        words += 1

        # Check word's spelling
        before = time.process_time()
        misspelled = not d.check(word)
        after = time.process_time()

        # Update benchmark
        time_check += after - before

        # Print word if misspelled
        if misspelled:
            print(word)
            misspellings += 1

        # Prepare for next word
        index, word = 0, ""

# Close file
file.close()

# Determine dictionary's size
before = time.process_time()
n = d.size()
after = time.process_time()

# Calculate time to determine dictionary's size
time_size = after - before

# Unload dictionary
before = time.process_time()
unloaded = d.unload()
after = time.process_time()

# Abort if dictionary not unloaded
if not unloaded:
    print("Could not load $dictionary.")
    sys.exit(1)

# Calculate time to determine dictionary's size
time_unload = after - before
```

```

# Report benchmarks
print(f"\nWORDS MISSPELLED:     {misspellings}")
print(f"WORDS IN DICTIONARY:  {n}")
print(f"WORDS IN TEXT:        {words}")
print(f"TIME IN load:         {time_load:.2f}")
print(f"TIME IN check:        {time_check:.2f}")
print(f"TIME IN size:         {time_size:.2f}")
print(f"TIME IN unload:       {time_unload:.2f}")
print(f"TOTAL TIME:           {time_load + time_check + time_size + time_unload:.2f}\n")

# Success
sys.exit(0)
```

```python
# Conditions and relational operators

from cs50 import get_int

# Prompt user for x
x = get_int("x: ")

# Prompt user for y
y = get_int("y: ")

# Compare x and y
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x is equal to y")
```

```python
# Opportunity for better design

print("cough")
print("cough")
print("cough")
```

```python
# Better design

for i in range(3):
    print("cough")
```

```python
# Abstraction


def main():
    for i in range(3):
        cough()


def cough():
    """Cough once"""
    print("cough")


if __name__ == "__main__":
    main()
```

```python
# Abstraction with parameterization


def main():
    cough(3)


def cough(n):
    """Cough some number of times"""
    for i in range(n):
        print("cough")


if __name__ == "__main__":
    main()
```

```python
# Floating-point arithmetic

from cs50 import get_float

# Prompt user for x
x = get_float("x: ")

# Prompt user for y
y = get_float("y: ")

# Perform division for user
print(f"{x} divided by {y} is {x / y}")
```

```python
#!/usr/bin/env python

print("hello, world")
print("hello, world")
```

```python
# Floating-point imprecision

print(f"{1/10:.55f}")
```

```python
# get_int and print

from cs50 import get_int

i = get_int("integer: ")
print(f"hello, {i}")
```

```python
# Integer arithmetic

from cs50 import get_int

# Prompt user for x
x = get_int("x: ")

# Prompt user for y
y = get_int("y: ")

# Perform arithmetic
print(f"{x} plus {x} is {x + y}")
print(f"{x} minus {y} is {x - y}")
print(f"{x} times {y} is {x * y}")
print(f"{x} truly divided by {y} is {x / y}")
print(f"{x} floor-divided by {y} is {x // y}")
print(f"remainder of {x} divided by {y} is {x % y}")
```

```python
# Logical operators

from cs50 import get_char

# Prompt user for answer
c = get_char("answer: ")

# Check answer
if c == "Y" or c == "y":
    print("yes")
elif c == "N" or c == "n":
    print("no")
```

```python
# Integer overflow

from time import sleep

# Iteratively double i
i = 1
while True:
    print(i)
    i *= 2
    sleep(1)
```

```python
# Remainder operation

from cs50 import get_int

# Prompt user for integer
n = get_int("n: ");

# Check parity of integer
if n % 2 == 0:
    print("even")
else:
    print("odd")
```

```python
# Abstraction and scope

from cs50 import get_int


def main():
    i = get_positive_int("positive integer, please: ")
    print(f"{i} is a positive integer")


def get_positive_int(prompt):
    """Prompt user for positive integer"""
    while True:
        n = get_int(prompt)
        if n > 0:
            break
    return n


if __name__ == "__main__":
    main()
```

```python
# Return value

from cs50 import get_int


def main():
    x = get_int("x: ")
    print(square(x))


def square(n):
    """Return square of n"""
    return n**2


if __name__ == "__main__":
    main()
```

```python
# Conditions and relational operators

from cs50 import get_int

# Prompt user for number
i = get_int("number: ")

# Check sign of number
if i < 0:
    print("negative")
elif i > 0:
    print("positive")
else:
    print("zero")
```

```python
# get_string and print

from cs50 import get_string

s = get_string("name: ")
print(f"hello, {s}")
```

```python
# Demonstrates format

from cs50 import get_string

s = get_string("name: ")
print("hello, {}".format(s))
```

```python
# Floating-point arithmetic

from cs50 import get_float

f = get_float("F: ")
c = 5 / 9 * (f - 32)
print(f"{c:.1f}")
```

```python
# Printing a command-line argument

from sys import argv

if len(argv) == 2:
    print(f"hello, {argv[1]}")
else:
    print("hello, world")
```

```python
# Printing command-line arguments

from sys import argv

for s in argv:
    print(s)
```

```python
# Printing characters in an array of strings

from sys import argv

for s in argv:
    for c in s:
        print(c)
    print()
```

```python
# Explicitly casts chars to ints

from cs50 import get_string

s = get_string("Name: ")
for c in s:
    print(f"{c} {ord(c)}")
```

```python
# Buggy example for help50

s = get_string("Name: ")
print(f"hello, {s}")
```

```python
# Capitalizes string using str method

from cs50 import get_string

s = get_string()
for c in s:
    print(c.upper(), end="")
print()
```

```python
# Exits with explicit value

import sys

if len(sys.argv) != 2:
    print("missing command-line argument")
    sys.exit(1)

print(f"hello, {argv[1]}")
sys.exit(0)
```

```python
# Prints four question marks

print("????")
```

```python
# Prints four question marks using a loop

for i in range(4):
    print("?", end="")
print()
```

```python
# Prints any number of question marks, as specified by user

from cs50 import get_int

n = get_int("Number: ")
for i in range(n):
    print("?", end="")
print()
```

```python
# Prints a positive number of question marks, as specified by user

from cs50 import get_int

# Prompt user for a positive number
while True:
    n = get_int("Positive number: ")
    if n > 0:
        break

# Print out that many bricks
for i in range(n):
    print("#")
```

```python
# Prints a square of bricks, sized as specified by user

from cs50 import get_int

# Prompt user for a positive number
while True:
    n = get_int("Positive number: ")
    if n > 0:
        break

# Print out this many rows
for i in range(n):

    # Print out this many columns
    for j in range(n):
        print("#", end="")
    print()
```

```python
# Prints string char by char

from cs50 import get_string

s = get_string()
if s:
    for c in s:
        print(c)
```

```python
# Prints string char by char, one per line

from cs50 import get_string

s = get_string("input: ")
print("output:");
for c in s:
    print(c)
```

```python
# Determines the length of a string

from cs50 import get_string

s = get_string("Name: ")
print(len(s))
```

```python
# Iterative binary search

import sys
from cs50 import get_string

# Names in a phone book
book = [
    "Chen",
    "Kernighan",
    "Leitner",
    "Lewis",
    "Malan",
    "Muller",
    "Seltzer",
    "Shieber",
    "Smith"]


def main():

    # Prompt user for name
    name = get_string("Name: ")

    # Search for name
    left, right = 0, len(book) - 1
    while left <= right:
        middle = (left + right) // 2
        if name == book[middle]:
            print(f"Calling {name}")
            sys.exit(0)
        elif name < book[middle]:
            right = middle - 1
        elif name > book[middle]:
            left = middle + 1
    print("Quitting")
    sys.exit(1)


if __name__ == "__main__":
    main()
```

```python
# Recursive binary search

from cs50 import get_string

# Names in a phone book
book = [
    "Chen",
    "Kernighan",
    "Leitner",
    "Lewis",
    "Malan",
    "Muller",
    "Seltzer",
    "Shieber",
    "Smith"]


def main():

    # Prompt user for name
    name = get_string("Name: ")

    # Search for name
    if search(name, book):
        print(f"Calling {name}")
    else:
        print("Quitting")


def search(name, names):
    """Search names for name"""

    # No more names to search
    if not names:
        return False

    # Look at middle
    middle = len(names) // 2
    if name == names[middle]:
        return True

    # Search left half
    elif name < names[middle]:
        return search(name, names[:middle])

    # Search right half
    elif name > names[middle]:
        return search(name, names[middle + 1:])


if __name__ == "__main__":
    main()
```

```python
# Extracts a user's initials

from cs50 import get_string

s = get_string("Name: ")
initials = ""
for c in s:
    if c.isupper():
        initials += c
print(initials)
```

```python
# Linear search

import sys
from cs50 import get_string

# Names in a phone book
book = [
    "Chen",
    "Kernighan",
    "Leitner",
    "Lewis",
    "Malan",
    "Muller",
    "Seltzer",
    "Shieber",
    "Smith"]

# Prompt user for name
name = get_string("Name: ");

# Search for name
if name in book:
    print(f"Calling {name}")
    sys.exit(0)
print("Quitting")
```

```python
# Sums a range of numbers iteratively


from cs50 import get_int


def main():
    while True:
        n = get_int("Positive integer: ")
        if n > 0:
            break
    answer = sigma(n)
    print(answer)


def sigma(m):
    """Return sum of 1 through m"""
    sum = 0
    for i in range(m + 1):
        sum += i
    return sum


if __name__ == "__main__":
    main()
```

```python
# Sums a range of numbers recursively


from cs50 import get_int


def main():
    while True:
        n = get_int("Positive integer: ")
        if n > 0:
            break
    answer = sigma(n)
    print(answer)


def sigma(m):
    """Return sum of 1 through m"""
    if m <= 0:
        return 0
    else:
        return m + sigma(m - 1)


if __name__ == "__main__":
    main()
```

```python
# Compares two strings for equality

from cs50 import get_string

# Get two strings
s = get_string("s: ")
t = get_string("t: ")

# Compare strings for equality
if s == t:
    print("same")
else:
    print("different")
```

```python
# Compares two strings for equality while checking for errors

import sys
from cs50 import get_string

# Get a string
s = get_string("s: ")
if s is None:
    sys.exit(1)

# Get another string
t = get_string("t: ")
if t is None:
    sys.exit(1)

# Compare strings for equality
if s == t:
    print("same")
else:
    print("different")
```

```python
# Compares two strings for equality while checking (succinctly) for errors

import sys
from cs50 import get_string

# Get a string
s = get_string("s: ")
if not s:
    sys.exit(1)

# Get another string
t = get_string("t: ")
if not t:
    sys.exit(1)

# Compare strings for equality
if s == t:
    print("same")
else:
    print("different")
```

```python
# Capitalizes a copy of a string while checking for errors

import sys
from cs50 import get_string

# Get a string
s = get_string("s: ")
if not s:
    sys.exit(1)

# Capitalize first letter in copy
t = s.capitalize()

# Print strings
print(f"s: {s}")
print(f"t: {t}")

sys.exit(0)
```

```python
# Fails to swap two integers


def main():
    x = 1
    y = 2

    print(f"x is {x}, y is {y}")
    swap(x, y)
    print(f"x is {x}, y is {y}")


def swap(a, b):
    tmp = a
    a = b
    b = tmp


if __name__ == "__main__":
    main()
```

```python
# Swaps two integers


def main():
    x = 1
    y = 2

    print(f"x is {x}, y is {y}")
    x, y = y, x
    print(f"x is {x}, y is {y}")


if __name__ == "__main__":
    main()
```

```python
# Implements a list of unique numbers

from cs50 import get_int

# Memory for numbers
numbers = []

# Prompt for numbers (until EOF)
while True:

    # Prompt for number
    number = get_int("number: ")

    # Check for EOF
    if not number:
        break

    # Check whether number is already in list
    if number not in numbers:

        # Add number to list
        numbers.append(number)

# Print numbers
print()
for number in numbers:
    print(number)
```

```python
# Demonstrates objects

from cs50 import get_string
from student import Student

# Space for students
students = []

# Prompt for students' names and dorms
for i in range(3):
    name = get_string("name: ")
    dorm = get_string("dorm: ")
    students.append(Student(name, dorm))

# Print students' names and dorms
for student in students:
    print(f"{student.name} is in {student.dorm}.")
```

```python
# Demonstrates file I/O

import csv
from cs50 import get_string
from student import Student

# Space for students
students = []

# Prompt for students' names and dorms
for i in range(3):
    name = get_string("name: ")
    dorm = get_string("dorm: ")
    students.append(Student(name, dorm))

with open("students.csv", "w") as file:
    writer = csv.writer(file)
    for student in students:
        writer.writerow((student.name, student.dorm))
```

```python
# Represents a student


class Student:
    def __init__(self, name, dorm):
        self.name = name
        self.dorm = dorm
```