# Deck 003
# Python – Strings ‘n’ Things

Dr. Hazel “twitch.tv/hazeldotzone” Campbell

Copyright 2026, Dr. Hazel Victoria Campbell, All Rights Reserved

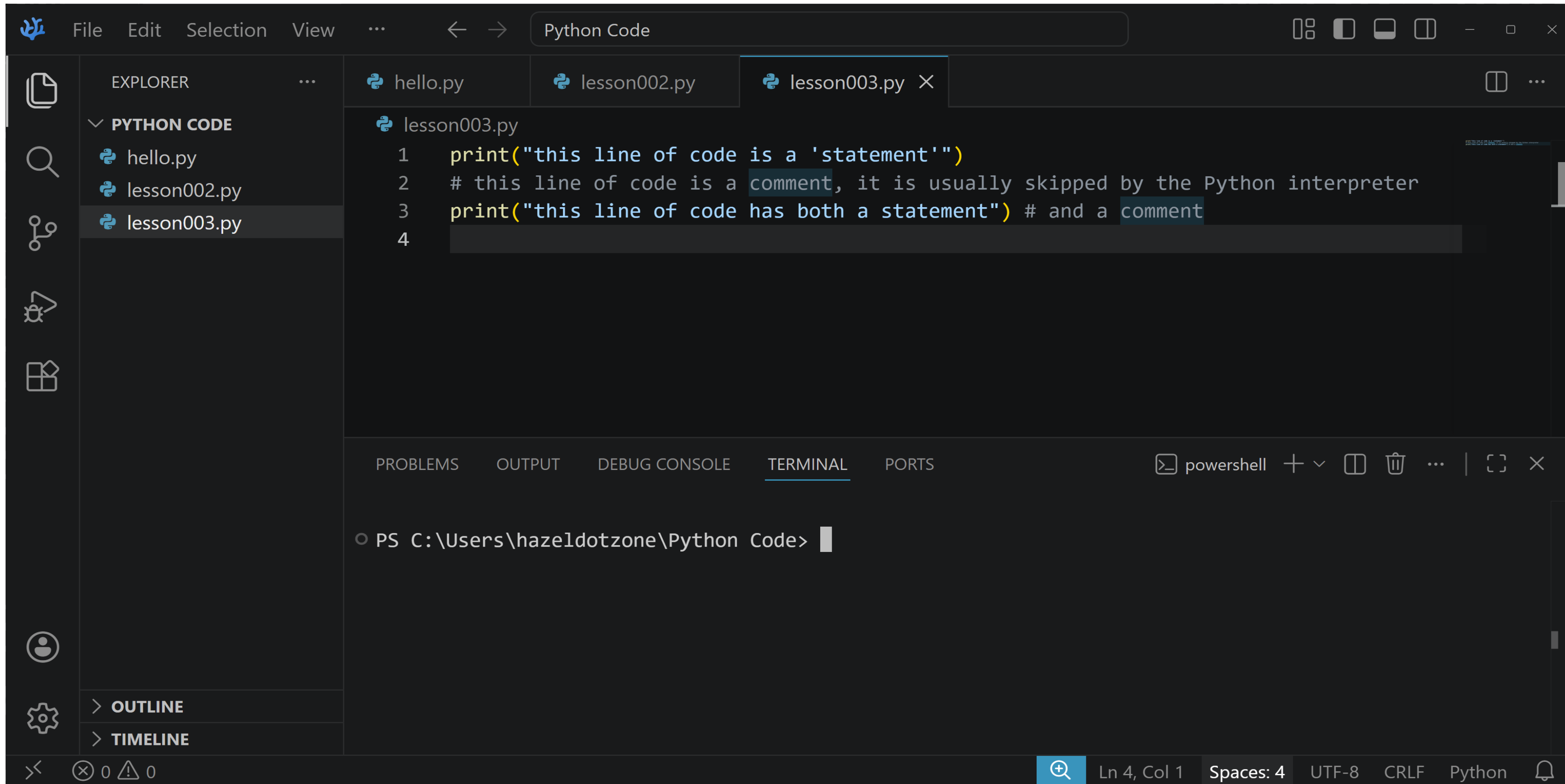
# Comments

#

- number sign
- hash
- octothorpe (it has 8 thorpes)


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Comments





Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Comments

Our Editor
helpfully colors
them for us!

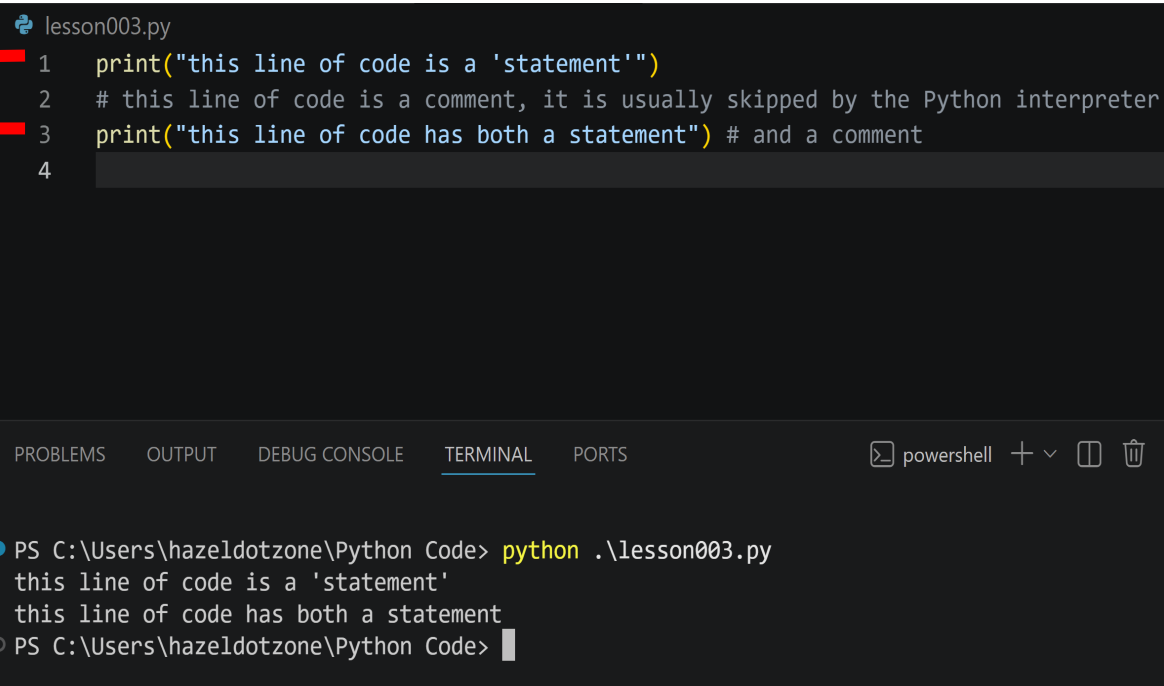
lesson003.py

```
1  print("this line of code is a 'statement'")
2  # this line of code is a comment, it is usually skipped by the Python interpreter
3  print("this line of code has both a statement") # and a comment
4
```

Python comments start with a # and end at the end of the line


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Comments





Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Comments

```
lesson003.py
1   print("this line of code is a 'statement'")
2   this line of code is a comment, it is usually skipped by the Python interpre
3   print("this line of code has both a statement") # and a comment
4
```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS    powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 2
    this line of code is a comment, it is usually skipped by the Python interpreter
         ^^^^
SyntaxError: invalid syntax
PS C:\Users\hazeldotzone\Python Code>
```

Python probably thinks there should be an operator where it’s pointing ^^^^


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Comments

```
1    print("this line of code is a 'statement'")
2    # this line of code is a comment, it is usually skipped by the Python interpreter
3    # print("this line of code has both a statement") # and a comment
4
```

The second print now no longer runs at all. It is not even checked for syntax.
Python comments are skipped right over.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
this line of code is a 'statement'
PS C:\Users\hazeldotzone\Python Code>
```

Be careful not to accidentally comment out Python code.
If that happens, it can cause mysterious problems.


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Comments

- A # starts a comment, except for when it doesn't

lesson003.py

```
1  print("this line of code is a 'statement'")
2  # this line of code is a comment, it is usually skipped by the Python interpreter
3  print("this line of code has both a statement") # and a comment
4  print("this octothorpe # is a part of the string literal") # but that one wasn't
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
this line of code is a 'statement'
this line of code has both a statement
this octothorpe # is a part of the string literal
PS C:\Users\hazeldotzone\Python Code>
```

A # inside of a string literal does NOT start a comment,
It is just another *character*
Inside the string literal


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Comments

- A # starts a comment, except for when it doesn't

```
lesson003.py
1  print("this line of code is a 'statement'")
2  # this line of code is a comment, it is usually skipped by the Python interpreter
3  print("this line of code has both a statement") # and a comment
4  print("this octothorpe # is a part of the string literal") # but that one wasn't
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
this line of code is a 'statement'
this line of code has both a statement
this octothorpe # is a part of the string literal
PS C:\Users\hazeldotzone\Python Code>
```

A # inside of a string literal does NOT start a comment,
It is just another *character*
Inside the string literal


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Aside: Characters

- A character is a number, letter, space, symbol, emoji, or generally anything you might type with your keyboard.

A # inside of a string literal
does NOT start a comment,
It is just another *character*
Inside the string literal


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# What is a line anyway?

lesson003.py

```
1   print("this line of code is a 'statement'")
2   # this line of code is a comment, it is usually skipped by the Python interpreter
3   print("this line of code has both a statement") # and a comment
4   print("this octothorpe # is a part of the string literal") # that one wasn't
5   print("but I can't put a line feed character
6    in a string literal")
7
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 5
    print("but I can't put a line feed character
          ^
SyntaxError: unterminated string literal (detected at line 5)
PS C:\Users\hazeldotzone\Python Code>
```

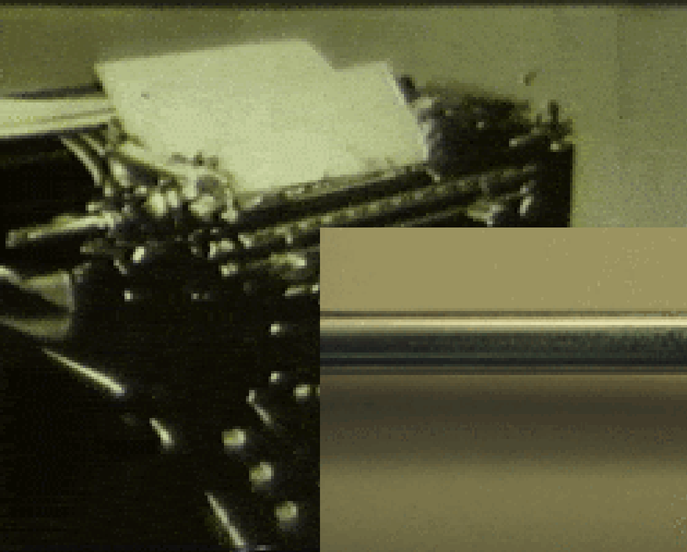
Ln 4, Col 44 Spaces: 4 UTF-8 CRLF

Windows defaults to using two characters, carriage return and line feed to start a new line in a text file, including in a source code file.

Other operating systems use just the line feed character.

Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# What is a line anyway?



Windows defaults to using two characters, carriage return and line feed to start a new line in a text file, including in a source code file.

Other operating systems use just the line feed character.


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Multi-line String Literals

- We can include a # in a string literal, but we cannot include a new line (line feed or line feed followed by a carriage return)

lesson003.py

```python
1  print("this line of code is a 'statement'")
2  # this line of code is a comment, it is usually skipped b
3  print("this line of code has both a statement") # and a c
4  print("this octothorpe # is a part of the string literal"
5  print("but I can't put a line feed character
6   in a string literal")
7
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 5
    print("but I can't put a line feed character
          ^
SyntaxError: unterminated string literal (detected at line 5)
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Multi-line String Literals

- Unless… we use three quote characters to make a multi-line string literal.

lesson003.py

```
1  print("this line of code is a 'statement'")
2  # this line of code is a comment, it is usually skipped b
3  print("this line of code has both a statement") # and a c
4  print("this octothorpe # is a part of the string literal"
5  print("but I can't put a line feed character
6   in a string literal")
7
8
```

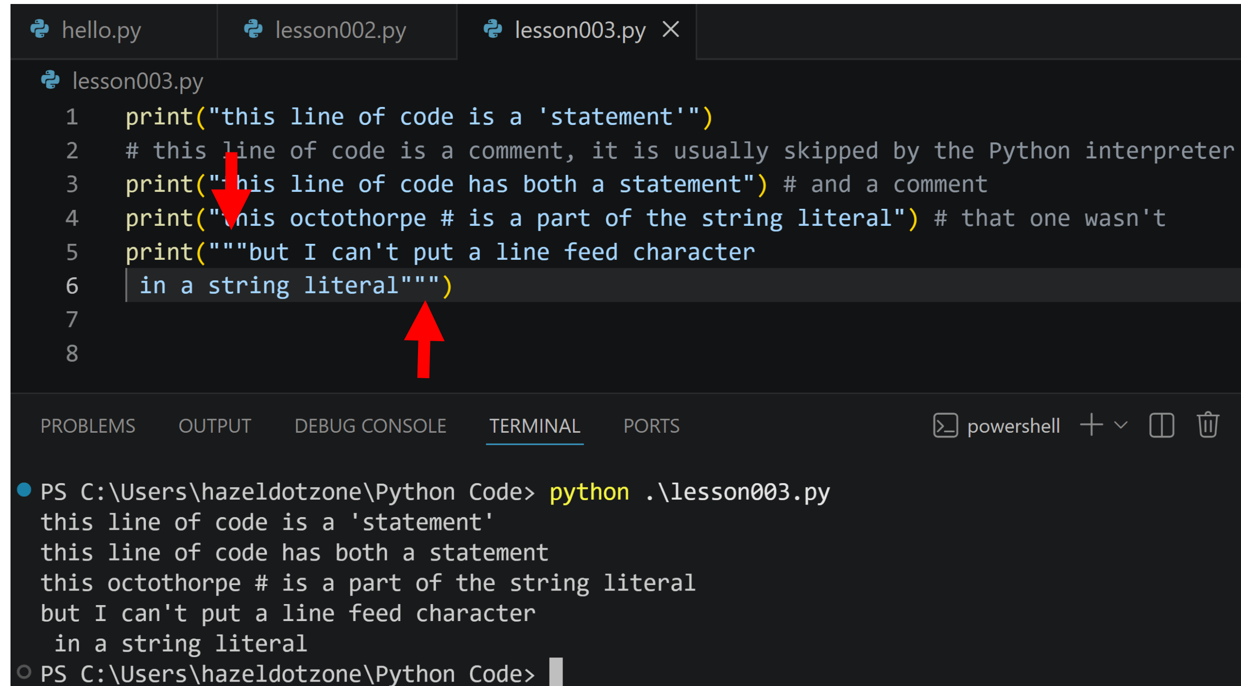
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 5
    print("but I can't put a line feed character
          ^
SyntaxError: unterminated string literal (detected at line 5)
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Multi-line String Literals

- Unless… we use three quote characters to make a multi-line string literal.





Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# More String Literals

- We can also use single quotes to make a string literal, as long as the start and end matches.

lesson003.py

```python
print("this line of code is a 'statement'")
# this line of code is a comment, it is usually skipped by the Python interpreter
print("this line of code has both a statement") # and a comment
print("this octothorpe # is a part of the string literal") # that one wasn't
print("""but I can't put a line feed character
 in a string literal""")
print('This string literal uses "single quotes"!')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
this line of code is a 'statement'
this line of code has both a statement
this octothorpe # is a part of the string literal
but I can't put a line feed character
 in a string literal
This string literal uses "single quotes"!
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# More String Literals

lesson003.py

```
1  print("this is a string literal")
2  print('this is a string literal')
3  print("this wont parse')
4
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 3
    print("this won't parse')
          ^
SyntaxError: unterminated string literal (detected at line 3)
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# More String Literals

lesson003.py

```
1    print("this is a string literal")
2    print('this is a string literal')
3    print("this wont parse')
4
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 3
    print("this won't parse')
          ^
SyntaxError: unterminated string literal (detected at line 3)
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

lesson003.py

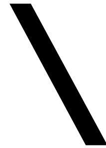
```python
print("using double quotes to delimit the string literal lets us easily use 'single quotes' in it")
print('using single quotes to delimit the string literal lets us easily use "double quotes" in it')
print("""using triple double quotes lets us easily use
line breaks
in the string literal""")
print('''We can also use
triple single quotes''')

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
using double quotes to delimit the string literal lets us easily use 'single quotes' in it
using single quotes to delimit the string literal lets us easily use "double quotes" in it
using triple double quotes lets us easily use
line breaks
in the string literal
We can also use
triple single quotes
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Aside: Slashes

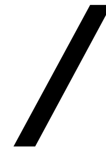


**Backslash**

Top leans backwards

Created so people could type /\ and \/

Invented by Bob Bemer at IBM



(forward) **slash**

Top leans forward


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# More String Literals

lesson003.py

```python
print("we can also get a double quote (even in a double quoted string) by writing \" backslash then the quote we want")
print('we can also get a double quote (even in a double quoted string) by writing \" backslash then the quote we want')
print("we can also get a single quote (even in a single quoted string) by writing \' backslash then the quote we want")
print('we can also get a single quote (even in a single quoted string) by writing \' backslash then the quote we want')
print("notice that the backslash itself is not included in the string created by the string literal")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
we can also get a double quote (even in a double quoted string) by writing " backslash then the quote we want
we can also get a double quote (even in a double quoted string) by writing " backslash then the quote we want
we can also get a single quote (even in a single quoted string) by writing ' backslash then the quote we want
we can also get a single quote (even in a single quoted string) by writing ' backslash then the quote we want
notice that the backslash itself is not included in the string created by the string literal
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# More String Literals

lesson003.py

```python
print("we can also get a double quote (even in a double quoted string) by writing \" backslash then the quote we want")
print('we can also get a double quote (even in a double quoted string) by writing \" backslash then the quote we want')
print("we can also get a single quote (even in a single quoted string) by writing \' backslash then the quote we want")
print('we can also get a single quote (even in a single quoted string) by writing \' backslash then the quote we want')
print("notice that the backslash itself is not included in the string created by the string literal")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORT

powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\l
we can also get a double quote (even in a double                          te we want
we can also get a double quote (even                                      he quote we want
we can also get a single quote (even in                                   the quote we want
we can also get a single quote (even in a                                 e quote we want
notice that the backslash itself is not i                                 al
PS C:\Users\hazeldotzone\Python
```

This is called an escape sequence. You will find these in any computer language.

# More String Literals

lesson003.py

```
1  print("we can also get a double quote (even in a double quoted string) by writing \" backslash then the quote we want")
2  print('we can also get a double quote (even in a double quoted string) by writing \" backslash then the quote we want')
3  print("we can also get a single quote (even in a single quoted string) by writing \' backslash then the quote we want")
4  print('we can also get a single quote (even in a single quoted string) by writing \' backslash then the quote we want')
5  print("notice that the backslash itself is not included in the string created by the string literal")
6
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORT

```
PS C:\Users\hazeldotzone\Python Code> python .\l
we can also get a double quote (even in a double
we can also get a double quote (even
we can also get a single quote (even in
we can also get a single quote (even in a                                e quote we want
notice that the backslash itself is not i                                al
PS C:\Users\hazeldotzone\Python
```

This is called an escape sequence. You will find these in any computer language.

Though, they don't always have the same syntax...

Also invented by Bob Bemer at IBM

# Escape Sequences: \’\”

Python sees \”
+ we are in a string literal
+ the next character is a escape sequence

evaluation

The Python `str` we get has the entire *escape sequence* replaced by a single *character: “*

Python sees \’
+ we are in a string literal
+ the next character is a escape sequence

evaluation

The Python `str` we get has the entire *escape sequence* replaced by a single *character: ‘*

```
"we can also get a double quote (even in a double quoted string) by writing \" backslash then the quote we want"
'we can also get a double quote (even in a double quoted string) by writing \" backslash then the quote we want'
"we can also get a single quote (even in a single quoted string) by writing \' backslash then the quote we want"
'we can also get a single quote (even in a single quoted string) by writing \' backslash then the quote we want'
```

 Copyright 2026 Dr Hazel Campbell https://hazel.zone/ 

Also invented by Bob Bemer at IBM

# Escape Sequences: `\r\n`

Python sees \
+ we are in a string literal
+ the next character is a escape sequence

evaluation

The Python `str` we get has the entire *escape sequence* replaced by a single *character*

Carriage Return    Line Feed

lesson003.py

```
1   print("another common escape sequence is \r\n for carriage-return line-feed")
2   print("or perhaps just \n for just line-feed")
3   print("""The alternative is our
4   triple-quoted string""")
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
another common escape sequence is
 for carriage-return line-feed
or perhaps just
 for just line-feed
The alternative is our
triple-quoted string
PS C:\Users\hazeldotzone\Python Code>
```

# `len()`

- It gets the length of something.
  - For strings it gets the length in characters.
- Calling it evaluates to an `int`


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# `len()`

- It gets the length of something.
  - For strings it gets the length in characters.
- Calling it evaluates to an `int`

lesson003.py

```python
print(len("a"))
print(len("ab"))
print(type(len("ab")))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
1
2
<class 'int'>
PS C:\Users\hazeldotzone\Python Code>
```

 Copyright 2026 Dr Hazel Campbell https://hazel.zone/ 

# Escape Sequences

```
1    print(len(""))   # prints 0 - the famous "zero length string"
2    print(len("a"))  # prints 1
3    print(len("ab")) # prints 2
4    print(len("\n")) # prints 1
5
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
0
1
2
1
PS C:\Users\hazeldotzone\Python Code>
```

Python sees \
+ we are in a string literal
+ the next character is a escape sequence

evaluation

The Python `str` we get has the entire *escape sequence* replaced by a single *character*

But how can we tell?


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Escape Sequences

lesson003.py

```
1   print(len(''))   # prints 0 - the famous "zero length string"
2   print(len('a'))  # prints 1
3   print(len('ab')) # prints 2
4   print(len('\n')) # prints 1
5
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
0
1
2
1
PS C:\Users\hazeldotzone\Python Code>
```

Remember!

In Python, we can use Double-quotes or Single-quotes for string literals, as long as both ends match!


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Escape Sequences: \\

lesson003.py

```
1    print("\")
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 1
    print("\")
          ^
SyntaxError: unterminated string literal (detected at line 1)
PS C:\Users\hazeldotzone\Python Code>
```

lesson003.py

```
1    print('\')
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 1
    print('\')
          ^
SyntaxError: unterminated string literal (detected at line 1)
PS C:\Users\hazeldotzone\Python Code>
```

Okay but what if we want an actual \ in our string?


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Escape Sequences: \\

```
lesson003.py
1    print('\\')
2    print("\\")
3    print(type("\\"))
4    print(len("\\"))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
\
\
<class 'str'>
1
PS C:\Users\hazeldotzone\Python Code>
```

Okay but what if we want an actual \ in our string?

Python evaluates the string literal "\\"

str with the value \


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Escape Sequences: \\

lesson003.py

```
1   print('a\\b')
2   print("a\\b")
3   print(type("a\\b"))
4   print(len("a\\b"))
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
a\b
a\b
<class 'str'>
3
PS C:\Users\hazeldotzone\Python Code>
```

Okay but what if we want an actual \ in our string?

Python evaluates the string literal "a\\b"

str with the value a\b



Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Escape Sequences: /???

lesson003.py

```
1  print('/')
2  print('//')
3  print('/r/n')
4  print(len('//'))
5
```

No error!

Both /s show up when we print them!

Nothing interesting happens if we do this!

It’s 2 characters long!

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
/
//
/r/n
2
PS C:\Users\hazeldotzone\Python Code>
```

(forward) slash does NOT start an escape sequence!

Only backslash does!


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Literal Concatenation

- Do not actually do this!
  - It only works for literals
  - It only works in Python and C
  - It's confusing to read

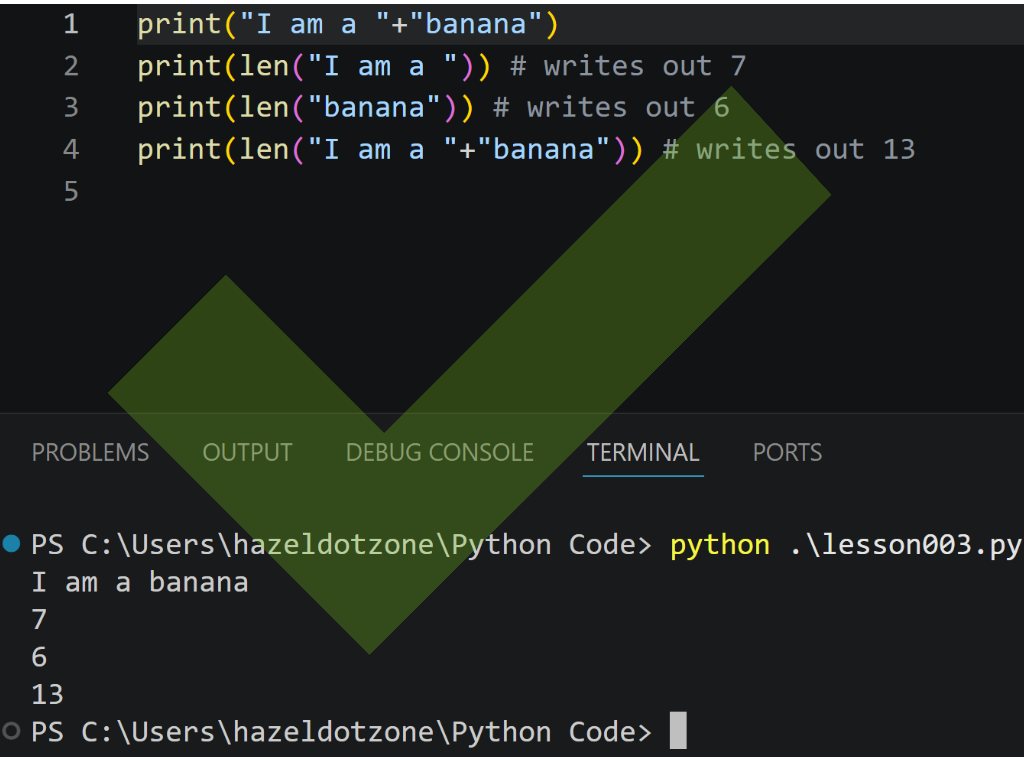
```
1  print("I am a " "banana")
2  print(len("I am a ")) # writes out 7
3  print(len("banana")) # writes out 6
4  print(len("I am a " "banana")) # writes out 13
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
I am a banana
7
6
13
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Literal Concatenation



Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Software Engineering Principle #1 Avoid non-*Idiomatic* Code

- Code which uses unusual or uncommon structures, syntax, features, patterns, etc.
- Keep code easy to read (for humans)

- Code is not just for communicating with the computer, but also for communicating with other programmers
  - That includes yourself (in the future)


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Literal Concatenation

Just because we won’t use it, does not mean we don’t need to be able to recognize it, unfortunately.

It’s easy to accidentally create code like this and unfortunately the Python parser won’t save us by giving us an error message!

Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Literal Concatenation

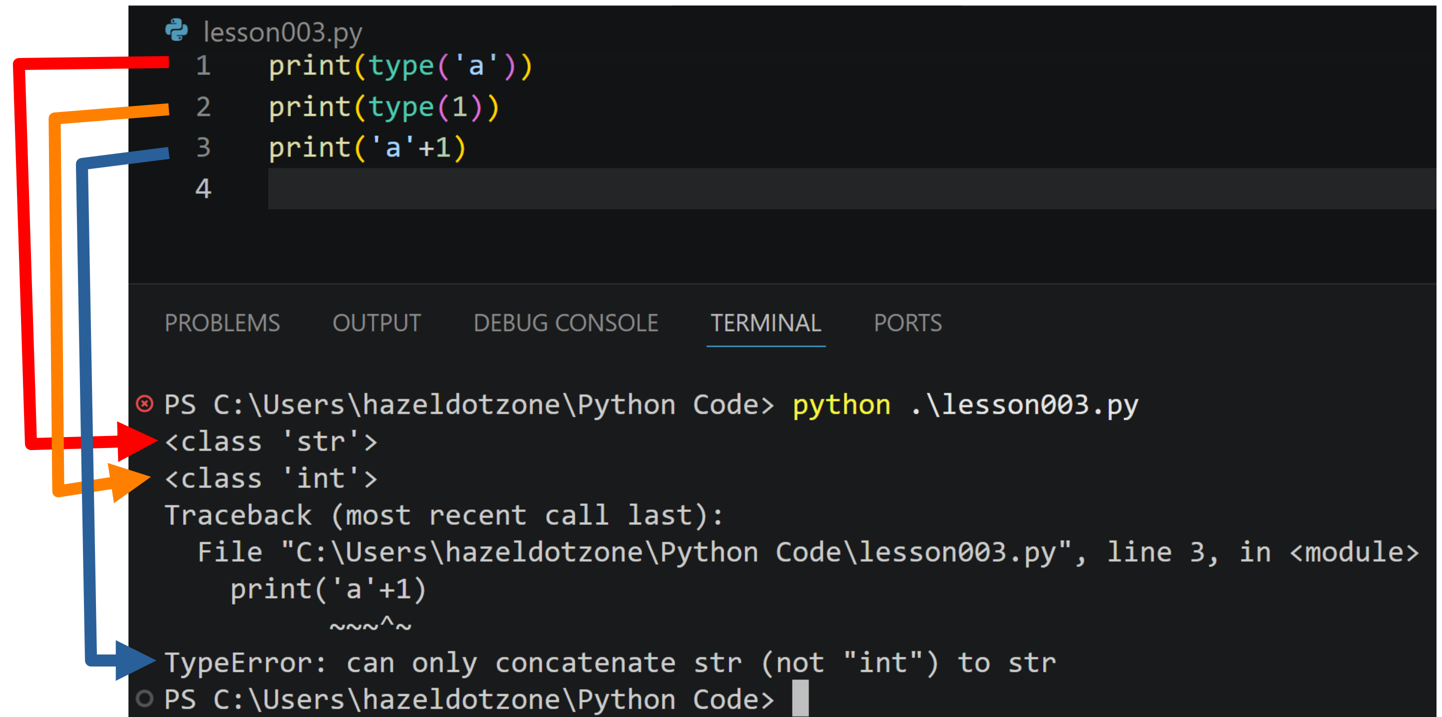
Just because we won't use it, does not mean we don't need to be able to recognize it, unfortunately.

It's easy to accidentally create code like this and unfortunately the Python parser won't save us by giving us an error message!

Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Types

- We can’t add two things of different types!




 Copyright 2026 Dr Hazel Campbell https://hazel.zone/ 

# Types

- We can’t add two things of different types!

Evaluates the left side of + its a `str`
So we must be doing concatenation
But the right side isn’t `str`
The right side is `int`

lesson003.py

```
1   print(type('a'))
2   print(type(1))
3   print('a'+1)
4
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<class 'str'>
<class 'int'>
Traceback (most recent call last):
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 3, in <module>
    print('a'+1)
          ~~~^~
TypeError: can only concatenate str (not "int") to str
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert something to a `str` with `str()`

lesson003.py

```python
print(type(1))
print(type(str(1)))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<class 'int'>
<class 'str'>
PS C:\Users\hazeldotzone\Python Code>
```

 Copyright 2026 Dr Hazel Campbell https://hazel.zone/ 

# Converting Types: `str`

- We can convert something to a `str` with `str()`

lesson003.py
```
1  print(type(1))
2  print(type(str(1)))
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

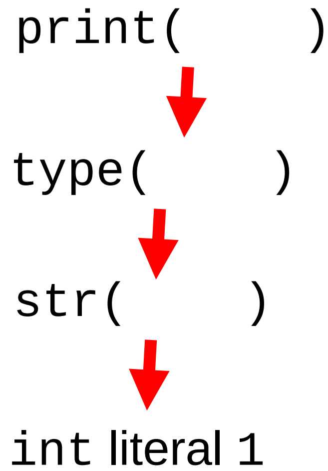
```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<class 'int'>
<class 'str'>
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert something to a `str` with `str()`

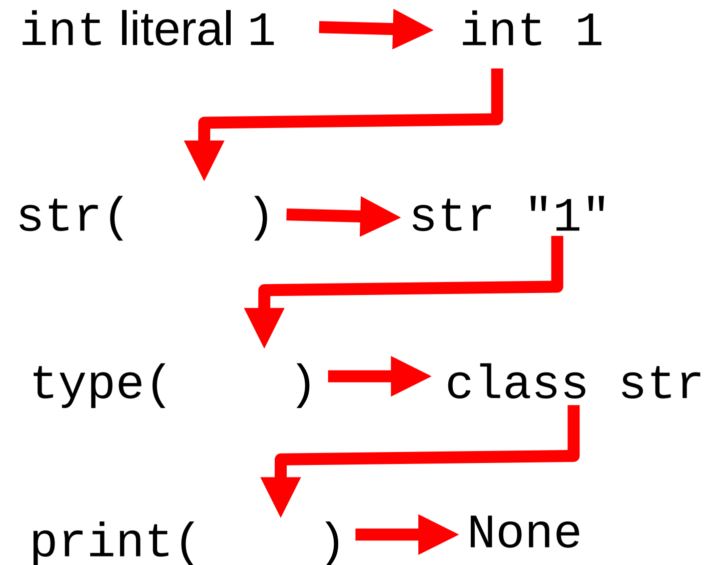
## Syntax Diagram




## Code

```
print(type(str(1)))
```

## Evaluation






Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert something to a `str` with `str()`
- Then we can concatentate it with another `str`

```
lesson003.py
1    print("a"+str(1))
2    print("a"+str(1+2))
3
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

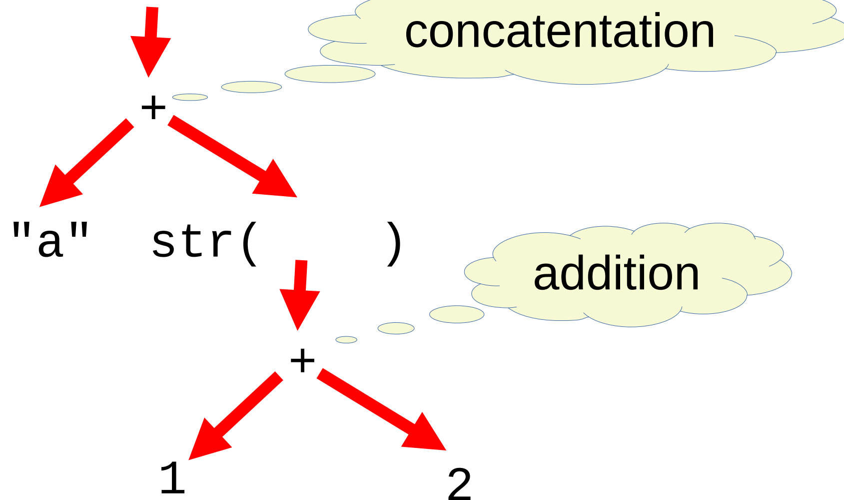
```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
a1
a3
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert something to a `str` with `str()`
- Then we can concatenate it with another `str`

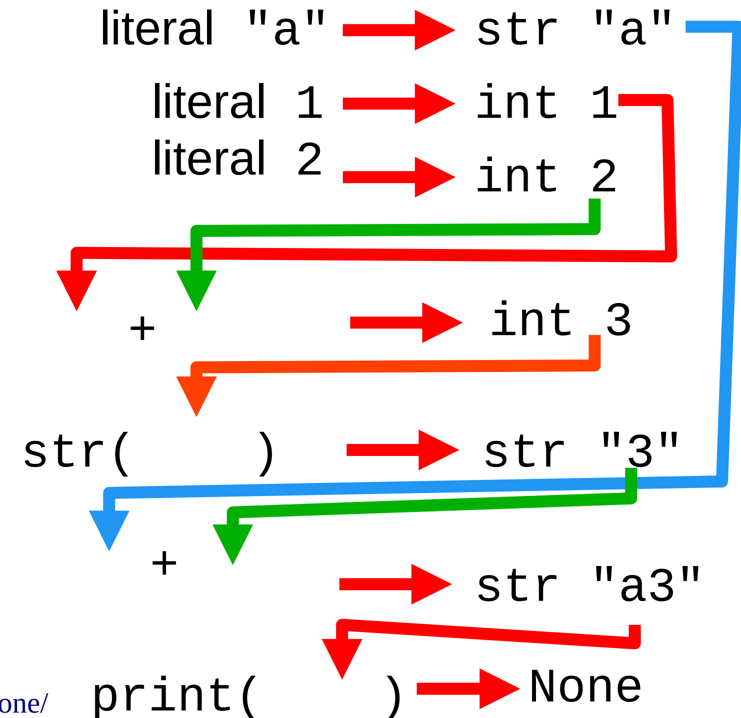
## Syntax Diagram




## Code

```
print("a"+str(1+2))
```

## Evaluation






Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert something to a `str` with `str()`
- Actually, these are all the same thing!
  - `type("a")`
  - `type(str(1))`
  - `str`

```
lesson003.py
1    print(type("a"))
2    print(type(str(1)))
3    print(str)
4
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<class 'str'>
<class 'str'>
<class 'str'>
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert something to a `str` with `str()`
- Actually, these are all the same thing!
  - `type("a")`
  - `type(str(1))`
  - `str`

lesson003.py

```python
print(type("a"))
print(type(str(1)))
print(str)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<class 'str'>
<class 'str'>
<class 'str'>
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert something to a `str` with `str()`
- Actually, these are all the same thing!
  - `type("a")`
  - `type(str(1))`
  - `str`
- Like really
  the same thing!

```
1    print(type(      str(1)))
2    print(type(type("a")(1)))
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<class 'str'>
<class 'str'>
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- What the heck is going on here:

```
print(type(type("a")(1)))
```

## Syntax Diagram

```
print(     )
 type(     )
type(    )(    )
   "a"     1
```

start of arguments delimiters


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

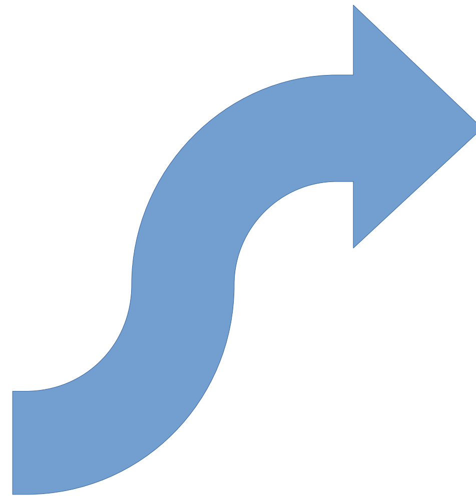
# Converting Types: `str`

- What the heck is going on here:

```
print(type(type("a")(1)))
```

## Syntax Diagram

```
print(     )
  type(     )
type(     )(     )
   "a"       1
```



## Evaluation

```
literal "a"  →  str "a"
type(    )  →  class str
literal 1  →  int 1
str(    )  →  str "1"
type(    )  →  class str
print(    )  →  None
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- What the heck is going on here: `print(type(type("a")(1)))`

## Syntax Diagram

```
print(      )
  type(    )
type(      )(      )
    "a"        1
```

These three functions are from identifiers program code

This function is from evaluating type("a")

## Evaluation

```
literal "a"  →  str "a"
type(   )    →  class str
literal 1    →  int 1
str(    )    →  str "1"
type(   )    →  class str
print(  )    →  None
```



Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert almost anything to str
  - Including functions...

lesson003.py

```python
print(print)
print(str(print))
print(len(str(print)))
print(len("<built-in function print>"))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<built-in function print>
<built-in function print>
25
25
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- We can convert almost anything to str
  - Including functions
  - and types...

```
lesson003.py
1   print(int)
2   print(str(int))
3   print(len(str(int)))
4   print(len("<class 'int'>"))
5
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<class 'int'>
<class 'int'>
13
13
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `str`

- Actually, print() just calls str() on anything you give it.

```
lesson003.py
1    print(str(123)+" is a "+str(type(123)))
2
3
```

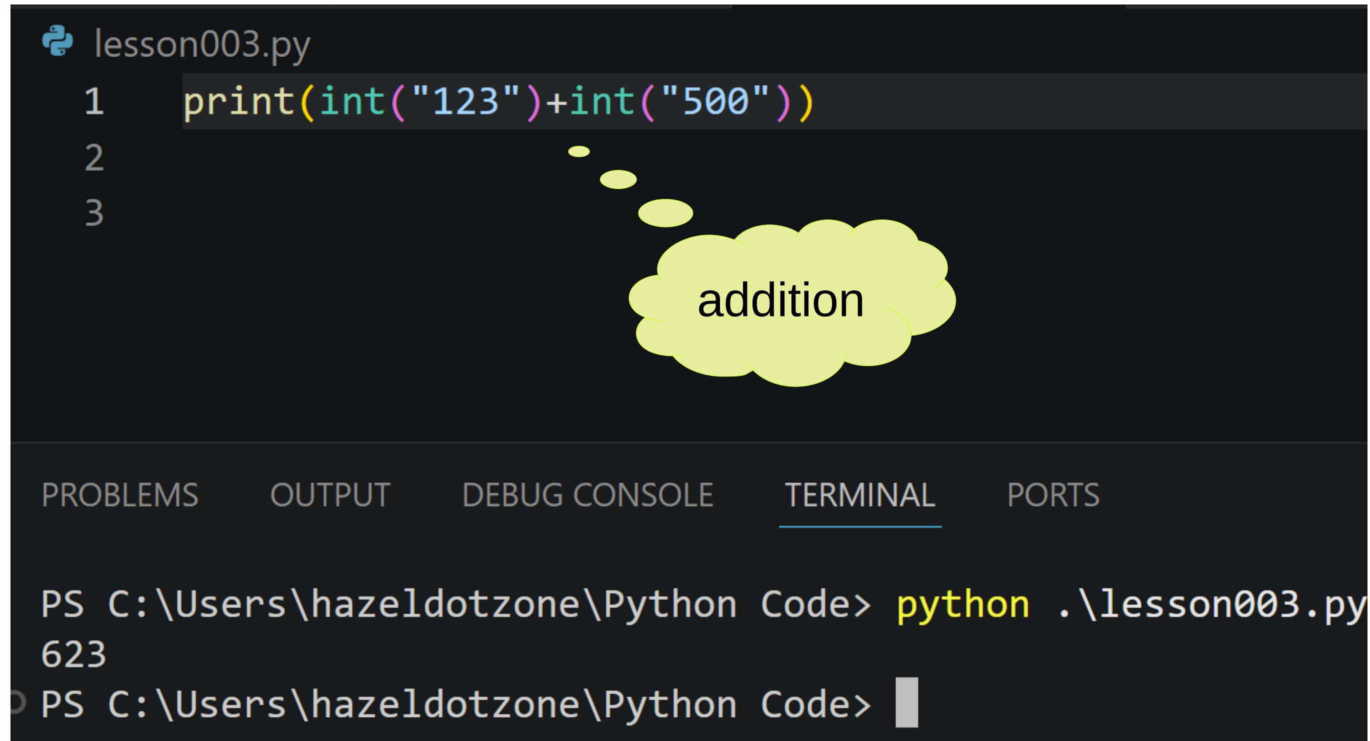
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
123 is a <class 'int'>
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `int`

- We can call `int(`on something`)` to convert it to an `int`





Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `int`

- We can call `int(`on something`)` to convert it to an `int`

lesson003.py

```
1    print(int("123")+int("500"))
2    print(type(int("123")))
3
4
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
623
<class 'int'>
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `int`

- We can call `int(on something)` to convert it to an `int`
- As long as it's something that makes sense to convert to an int

```
lesson003.py
1    print(int(""))
2
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
Traceback (most recent call last):
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 1, in <module>
    print(int(""))
          ^^^^^^^
ValueError: invalid literal for int() with base 10: ''
PS C:\Users\hazeldotzone\Python Code>
```

Nope, we can't convert the empty string
to an `int`.


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `int`

- We can call `int(`on something`)` to convert it to an `int`
- As long as it's something that makes sense to convert to an int

lesson003.py

```
1    print(int("a"))
2
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
Traceback (most recent call last):
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 1, in <module>
    print(int("a"))
          ^^^^^^^^
ValueError: invalid literal for int() with base 10: 'a'
PS C:\Users\hazeldotzone\Python Code>
```

Nope, we can't convert the string

a

to an `int`.


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `int`

- We can call `int(on something)` to convert it to an `int`
- As long as it's something that makes sense to convert to an int

lesson003.py

```python
print(int("10a"))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
Traceback (most recent call last):
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 1, in <module>
    print(int("a"))
          ^^^^^^^^
ValueError: invalid literal for int() with base 10: 'a'
PS C:\Users\hazeldotzone\Python Code>
```

Nope, we can't convert the string

`10a`

to an `int`.



Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Converting Types: `int`

- We can call `int(on something)` to convert it to an `int`
- As long as it's something that makes sense to convert to an int

lesson003.py

```
1    print(int("-10"))
2
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
-10
```

Of course…
in this example, we are converting a `str` to an `int` then print calls `str` on it to convert it right back to a `str`!


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

- We can use the input function to read input from the terminal.
- In this situation it will read from the terminal.
- You get to write your own prompt string!

```python
1  print(input)
2  print(input("What's your name? "))
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<built-in function input>
What's your name? Hazel
Hazel
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

- We can use the input function to read input from the terminal.
- In this situation it will read from the terminal.
- Here’s a simple interactive program:

lesson003.py

```python
print(input)
print("Hello "+input("What's your name? ")+"!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

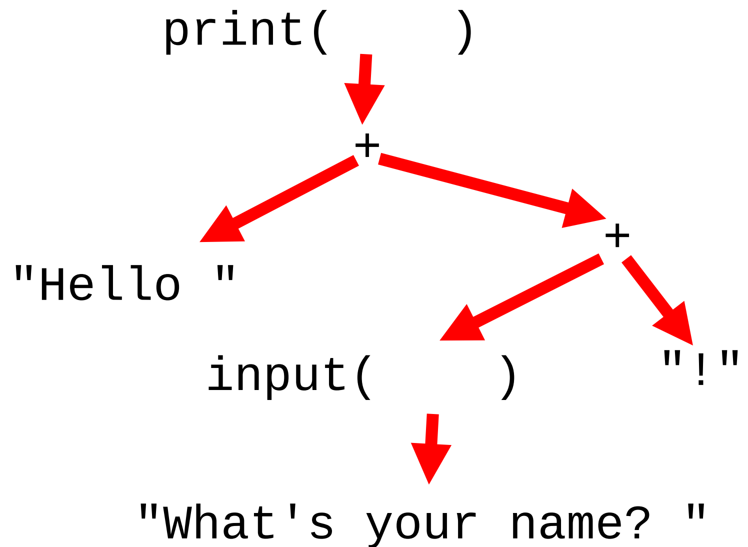
```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
<built-in function input>
What's your name? Hazel
Hello Hazel!
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

```
print("Hello "+input("What's your name? ")+"!")
```

## Syntax Diagram

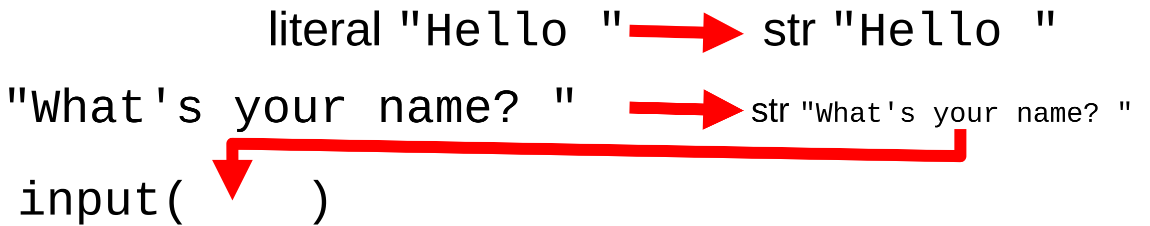


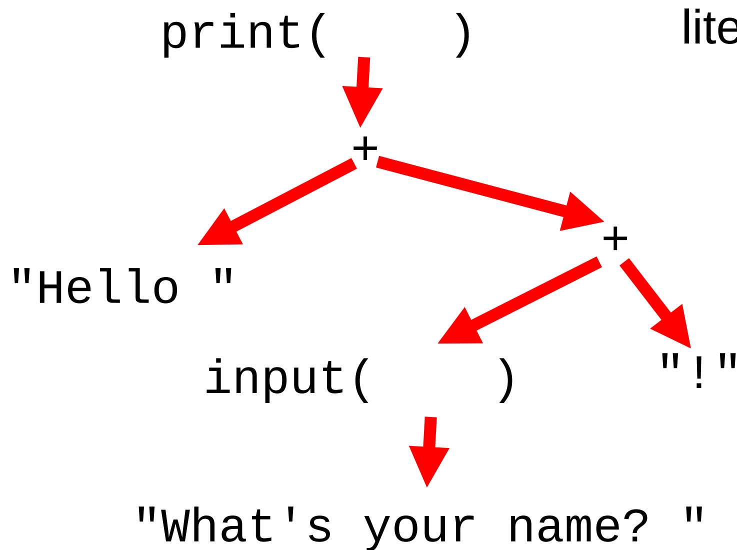


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

```
print("Hello "+input("What's your name? ")+"!")
```



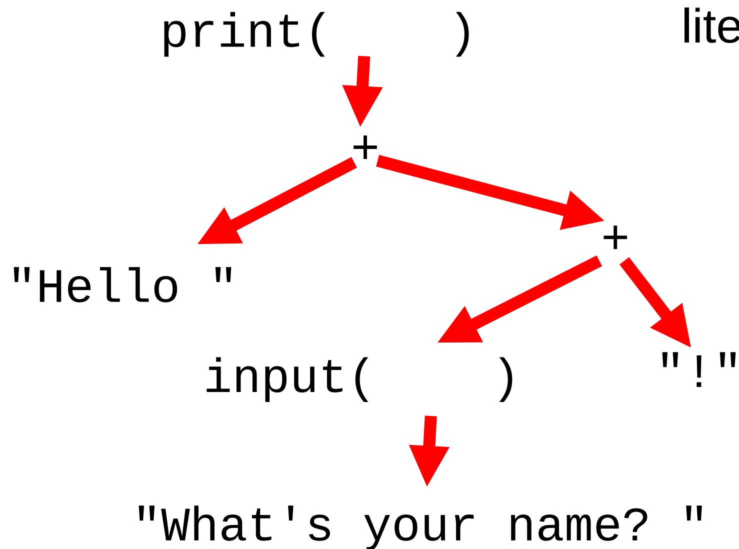



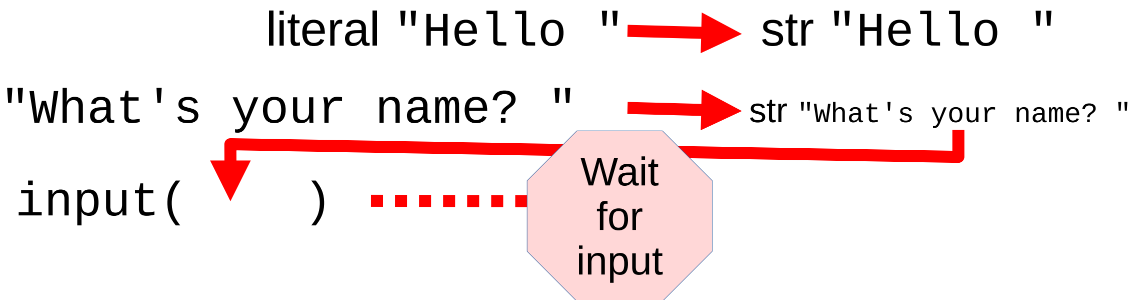


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

```
print("Hello "+input("What's your name? ")+"!")
```



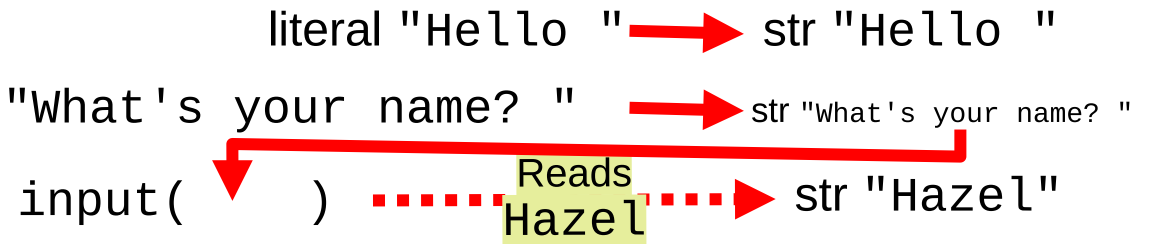





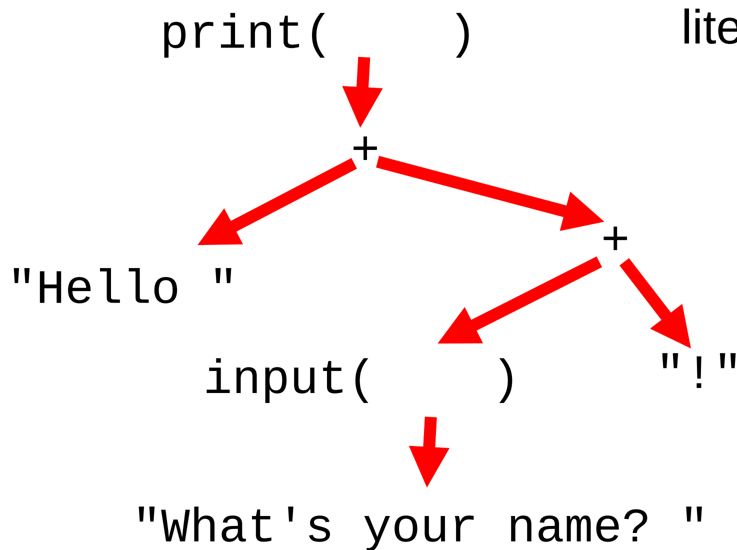
Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

```
print("Hello "+input("What's your name? ")+"!")
```




## Syntax Diagram

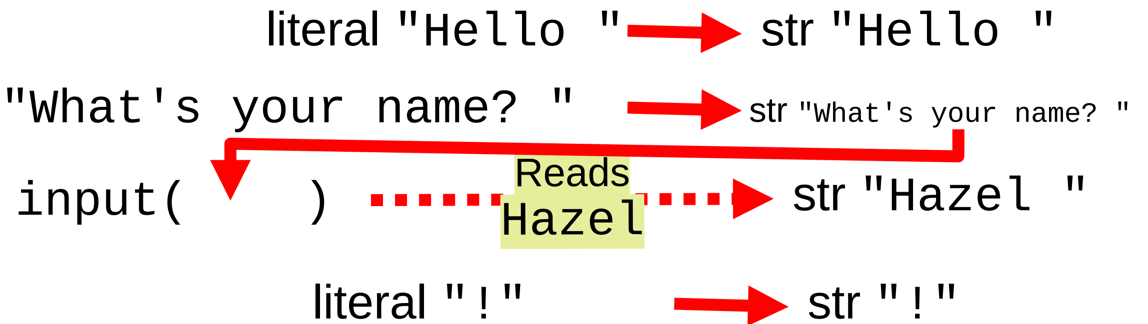


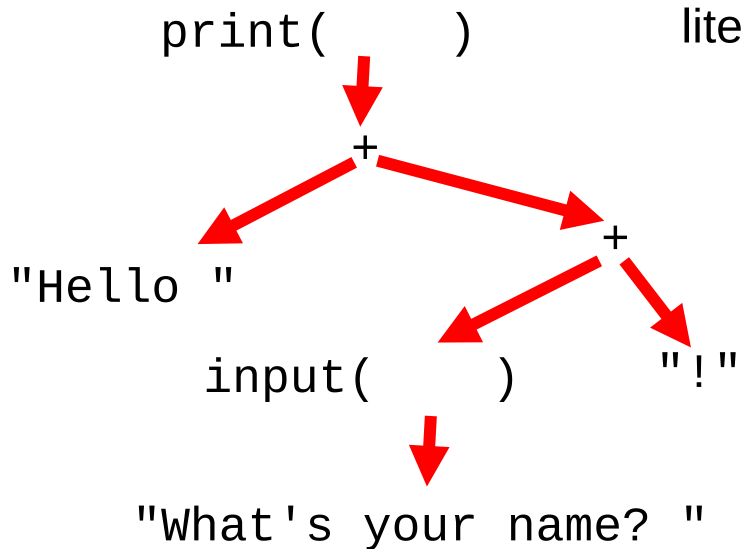


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

```
print("Hello "+input("What's your name? ")+"!")
```



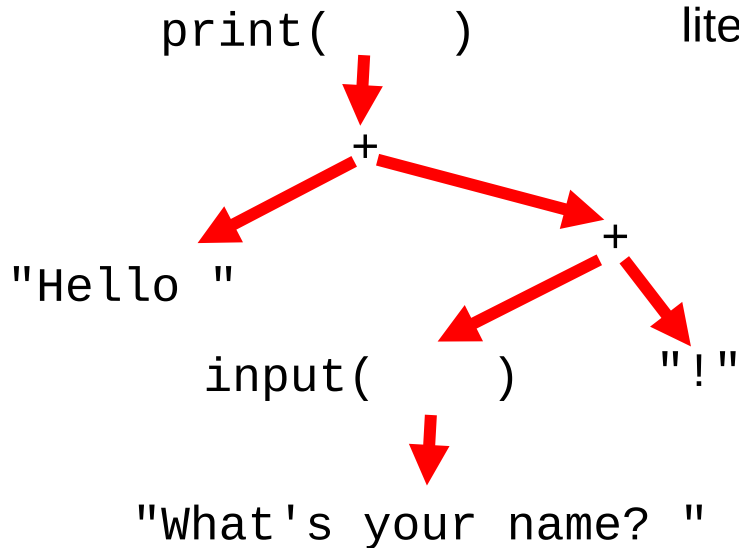



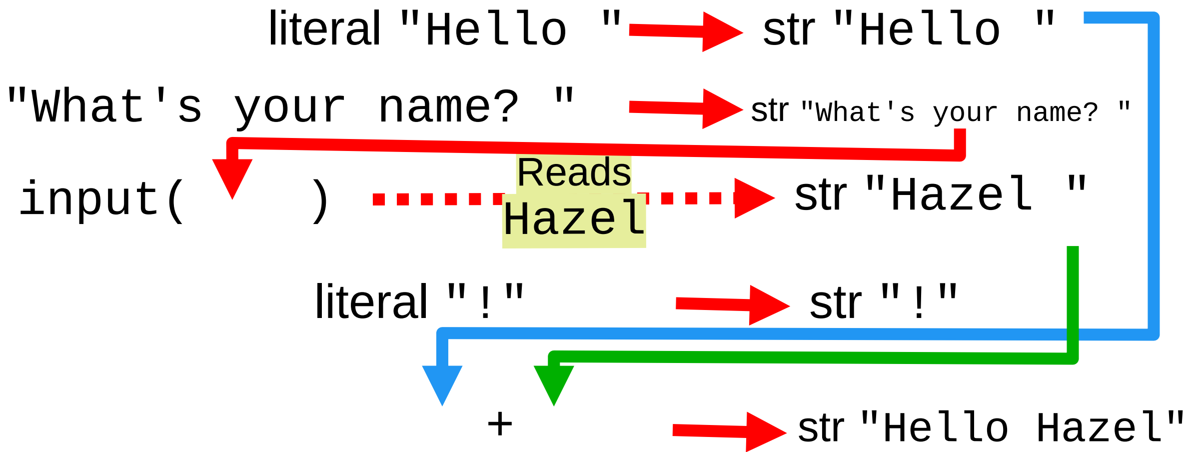


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

```
print("Hello "+input("What's your name? ")+"!")
```



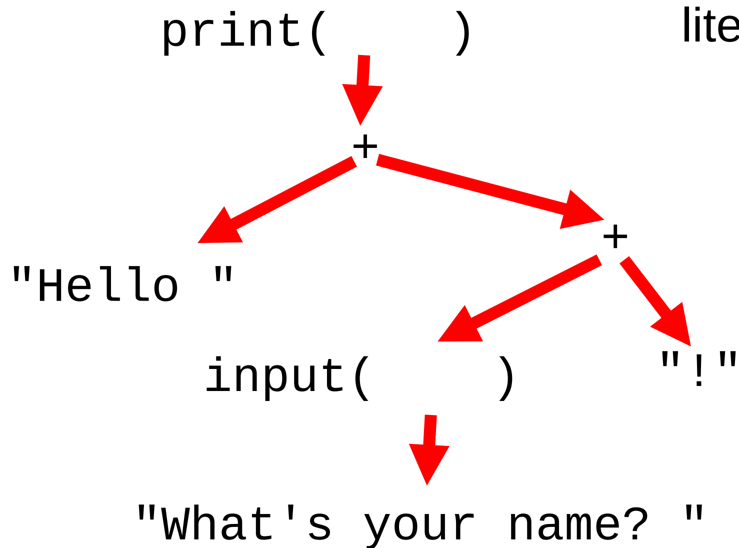



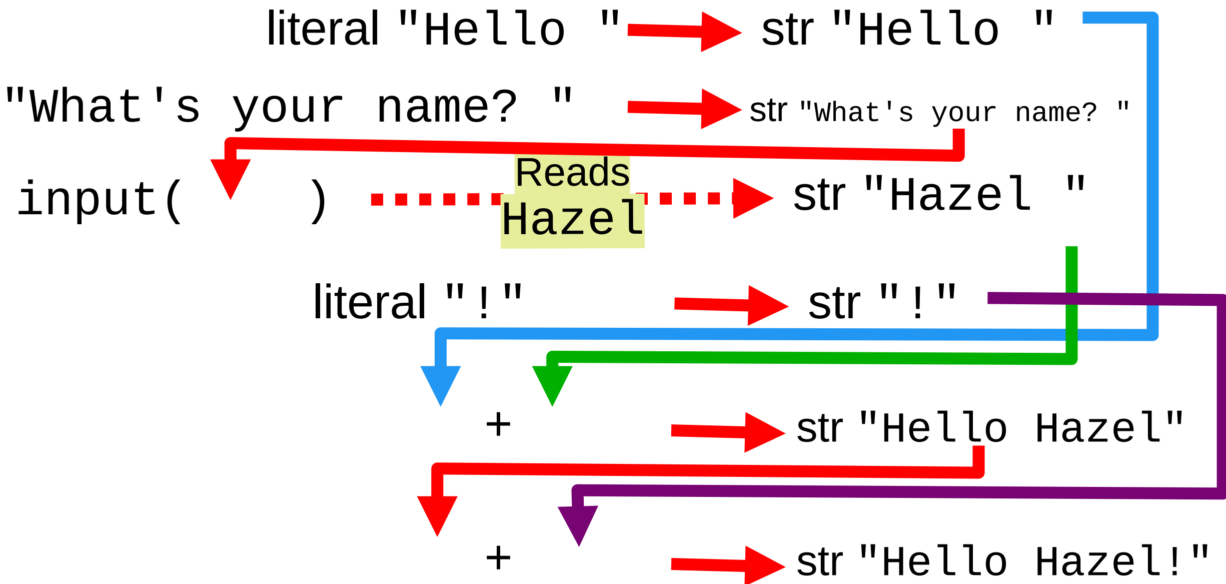


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

```
print("Hello "+input("What's your name? ")+"!")
```



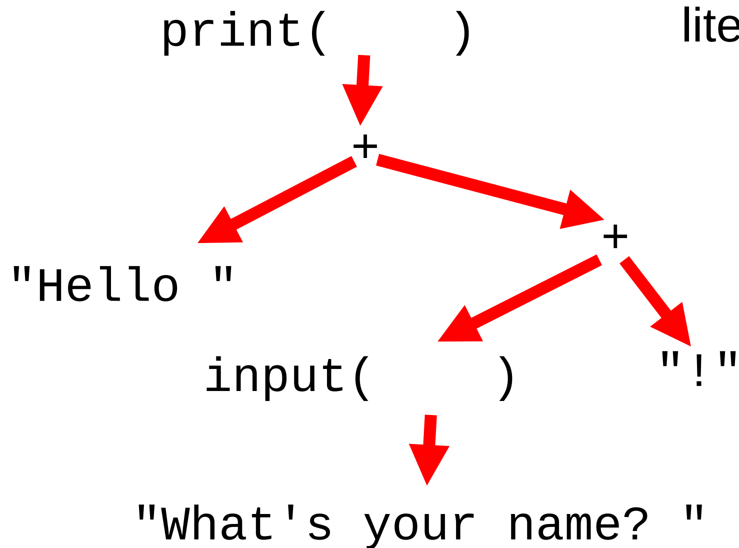



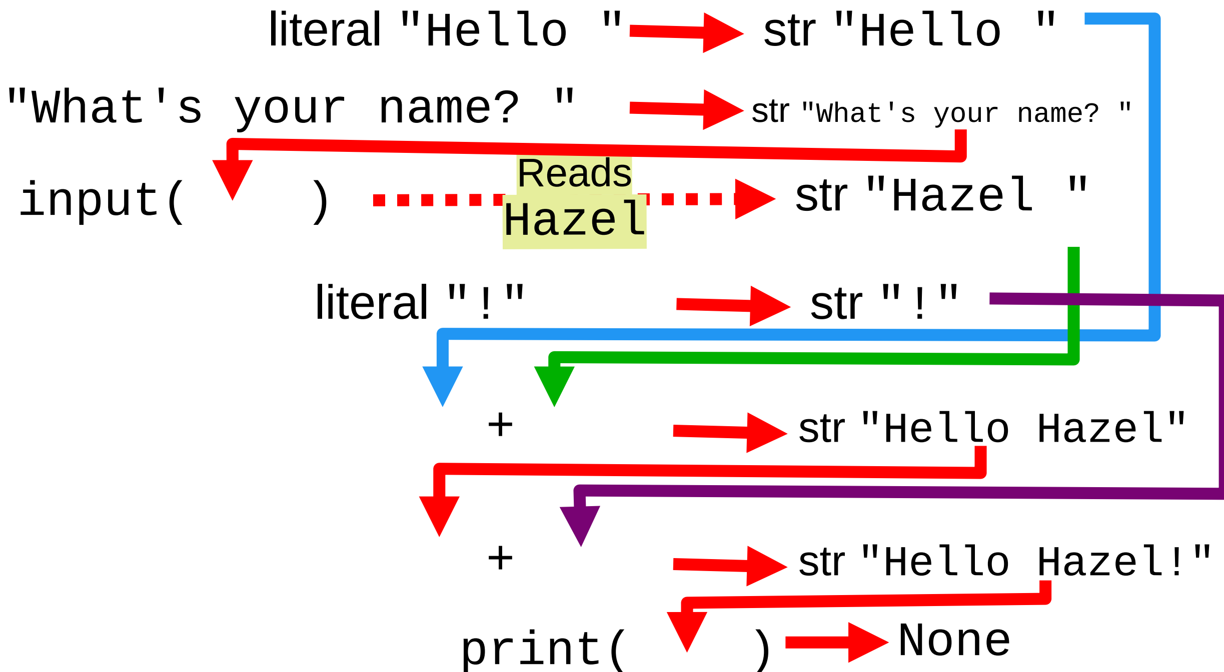


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

```
print("Hello "+input("What's your name? ")+"!")
```





Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

- Here’s a simple interactive program to add one hundred to a number:

lesson003.py

```
1    print(100+int(input("Enter a number: ")))
2
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
Enter a number: 555
655
PS C:\Users\hazeldotzone\Python Code>
```


Copyright 2026 Dr Hazel Campbell https://hazel.zone/

# Reading Input: `input()`

- Here’s a simple interactive program to add one hundred to a number:

lesson003.py

```python
print(100+int(input("Enter a number: ")))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
Enter a number: 555
655
```

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson003.py
Enter a number: of course, it doesn't work without a number
Traceback (most recent call last):
  File "C:\Users\hazeldotzone\Python Code\lesson003.py", line 1, in <module>
    print(100+int(input("Enter a number: ")))
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: "of course, it doesn't work without a number"
PS C:\Users\hazeldotzone\Python Code>
```

Copyright 2026 Dr Hazel Campbell https://hazel.zone/