Deck for Lesson 006
Python – Dots, Named Arguments

Dr. Hazel “twitch.tv/hazeldotzone” Campbell

The Build-A-String Workshop

- Making strings with some information in them is a common task when programming:

Line Length

- First of all, this line is way too long.

Line Length

- Generally, to improve readability, you should limit the line length to around 72-79 characters

Line Length

- Generally, to improve readability, you should limit the line length to around 72-79 characters
-

Line Length

- Generally, to improve readability, you should limit the line length to around 72-79 characters
- This has been the standard since 1928
- That is the standard for Python
- Some projects or organizations may have their own style guides such as 90, 100, 120 characters
- Nobody really wants to go over ~140

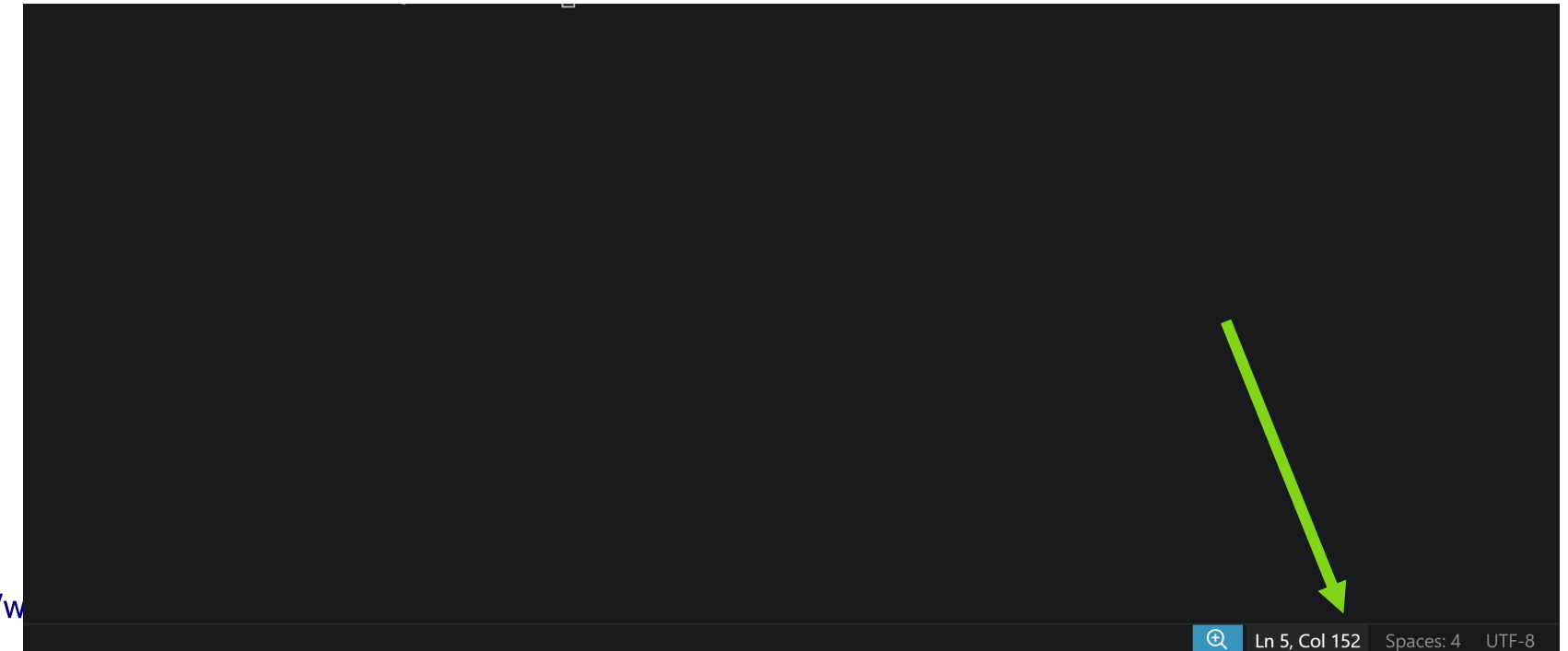
Line Length

- Generally, to improve readability, you should limit the line length to around 72-79 characters
- This is purely for readability. No one will really care if you go over by some characters, sometimes.

Line Length

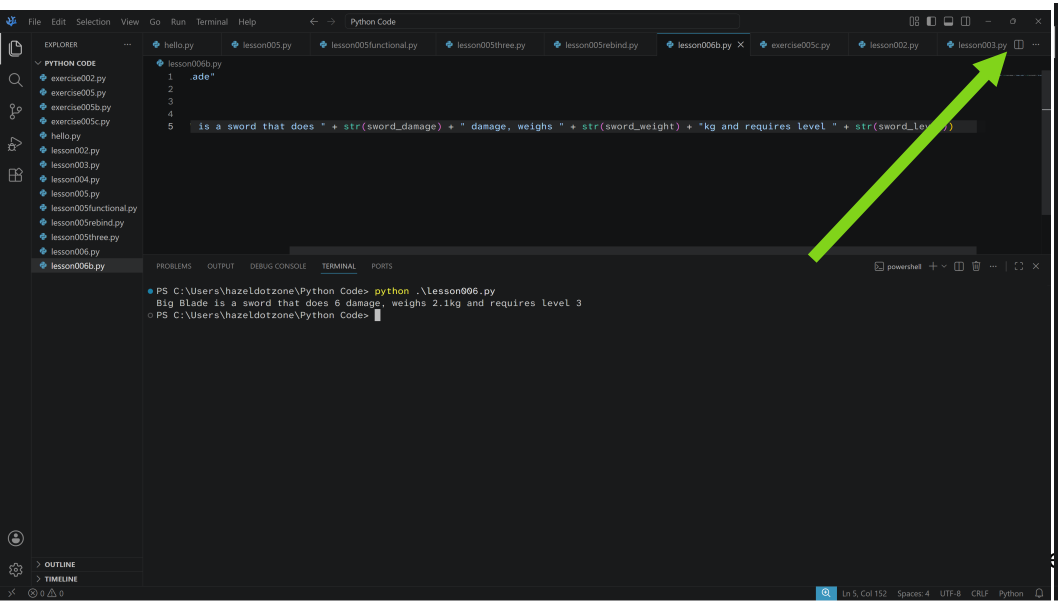
```
4
5   r(sword_damage) + " damage, weighs " + str(sword_weight) + "kg and requires level " + str(sword_level))
```

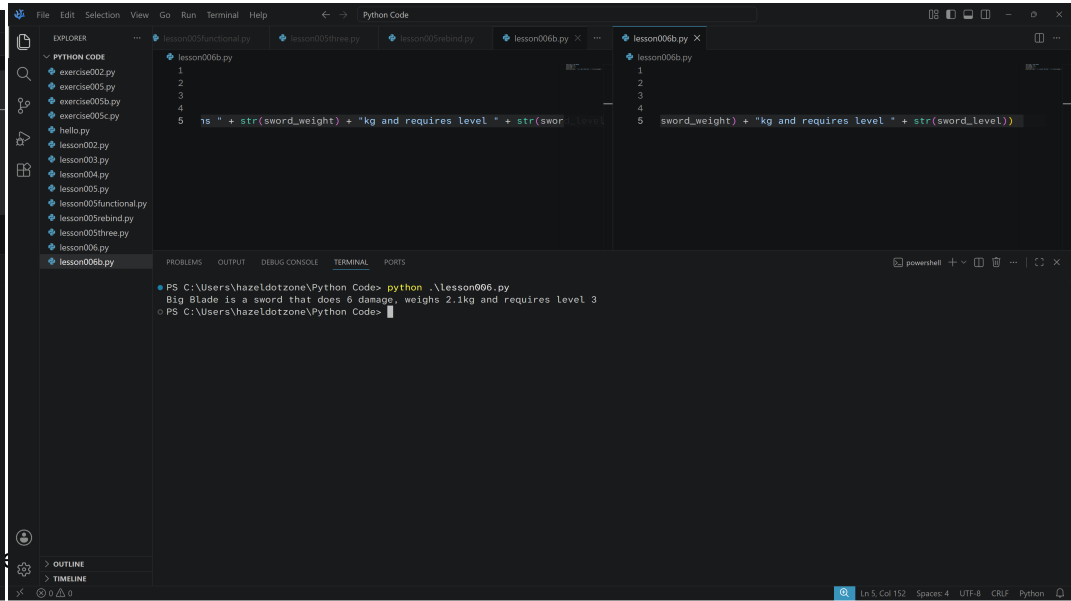
- However, 152 is generally excessive and hard to read.



Line Length

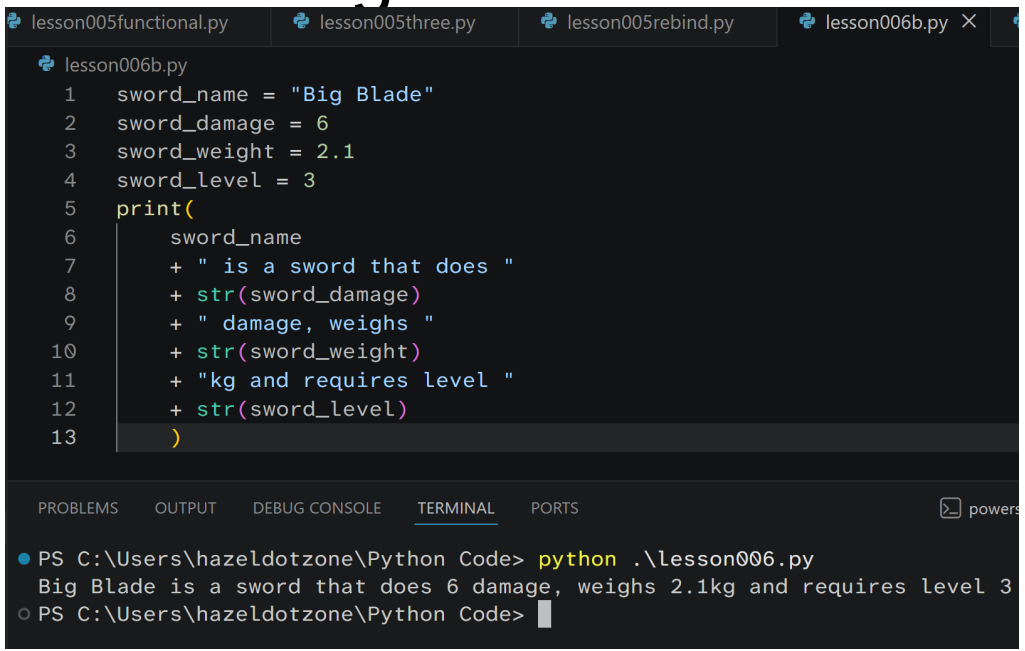
- Even programmers who have a large monitor and small font will usually use the space for other things such as two editors side by side





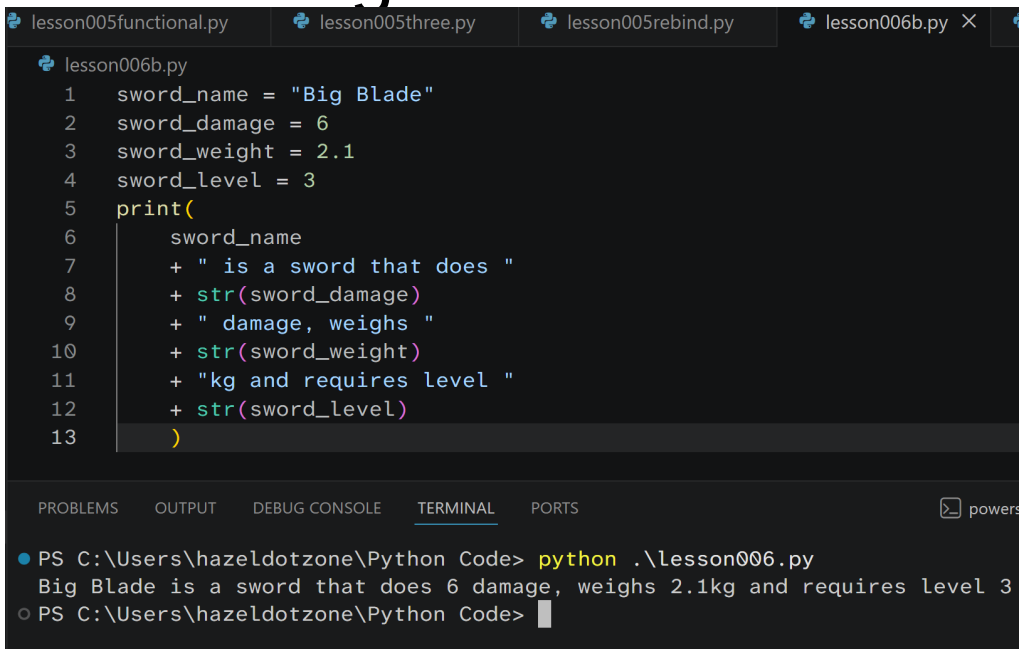
Line Length

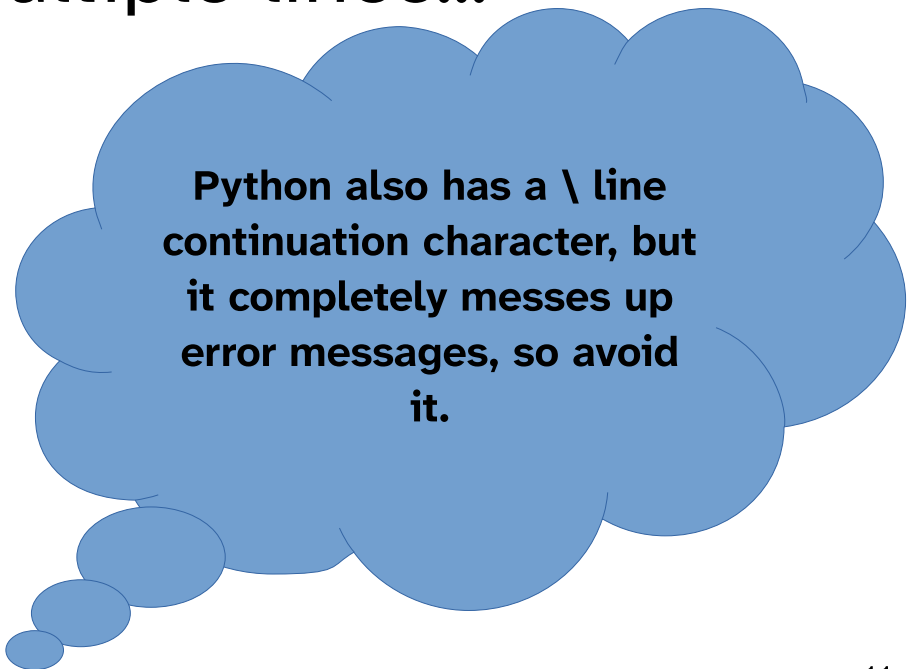
- We can always break up delimited expressions and arguments across multiple lines...



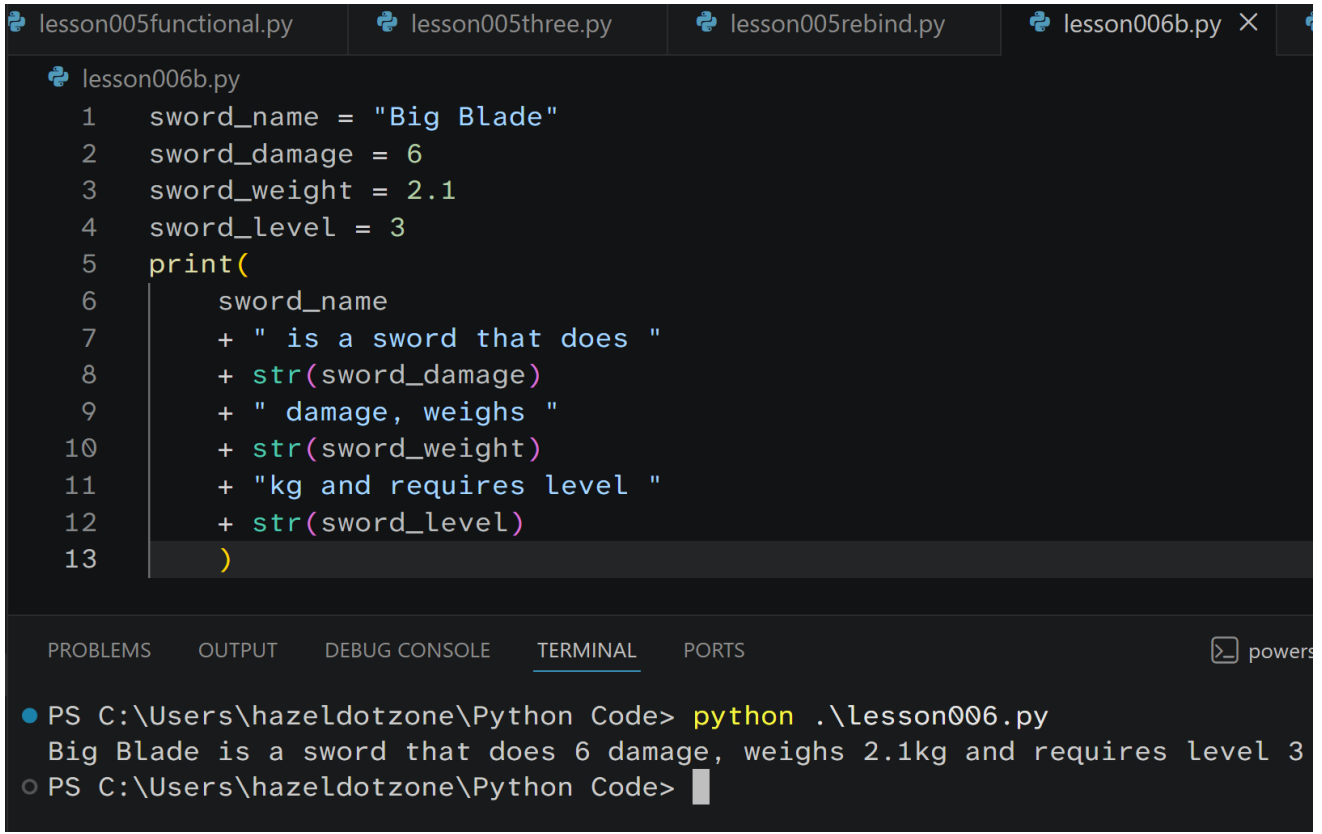
Line Length

- We can always break up delimited expressions and arguments across multiple lines...



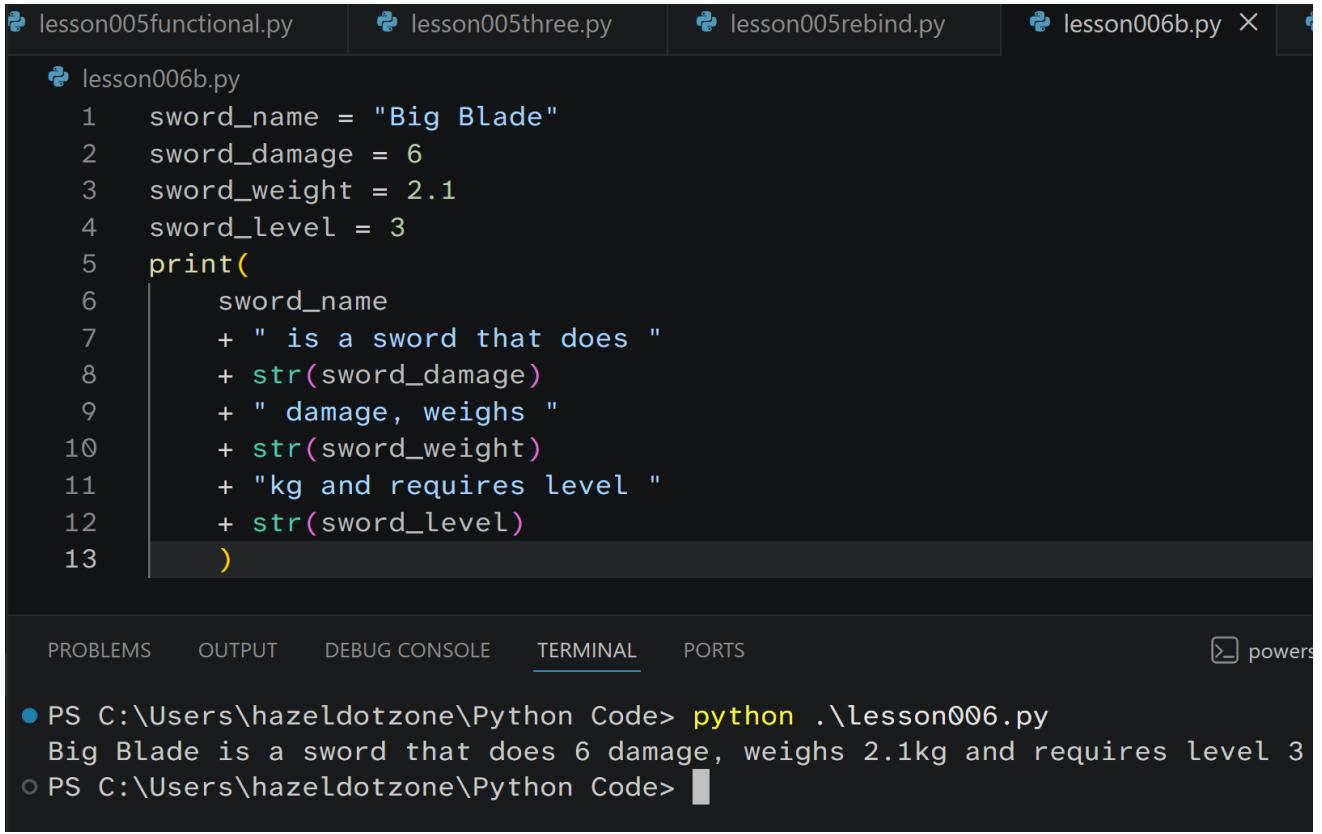


This is still cumbersome and annoying!



- "There's got to be a better way!"

This is still cumbersome and annoying!



- "There's got to be a better way!"

We can give multiple arguments to print()

lesson006c.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6
3   sword_weight = 2.1
4   sword_level = 3
5   print(
6       sword_name,
7       "is a sword that does",
8       str(sword_damage),
9       "damage, weighs",
10      str(sword_weight),
11      "kg and requires level",
12      str(sword_level)
13      )
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powersh

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006.py
Big Blade is a sword that does 6 damage, weighs 2.1kg and requires level 3
PS C:\Users\hazeldotzone\Python Code> python .\lesson006c.py
Big Blade is a sword that does 6 damage, weighs 2.1 kg and requires level 3
PS C:\Users\hazeldotzone\Python Code>
```

- We can give multiple arguments to print.

We can give multiple arguments to print()

lesson006c.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6
3   sword_weight = 2.1
4   sword_level = 3
5   print(
6       sword_name,
7       "is a sword that does",
8       sword_damage,
9       "damage, weighs",
10      sword_weight,
11      "kg and requires level",
12      sword_level
13      )
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershe

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006c.py
Big Blade is a sword that does 6 damage, weighs 2.1 kg and requires level 3
PS C:\Users\hazeldotzone\Python Code>
```

- We can give multiple arguments to print.
- It automatically calls str() on all of its arguments in case we didn't.

We can give multiple arguments to print()

lesson006c.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6
3   sword_weight = 2.1
4   sword_level = 3
5   print(
6       sword_name,
7       "is a sword that does",
8       sword_damage,
9       "damage, weighs",
10      sword_weight,
11      "kg and requires level",
12      sword_level
13      )
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershe

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006c.py
Big Blade is a sword that does 6 damage, weighs 2.1 kg and requires level 3
PS C:\Users\hazeldotzone\Python Code>
```

- We can give multiple arguments to print.
- It automatically calls str() on all of its arguments in case we didn't.
- It automatically adds spaces between things.

We can give multiple arguments to print()

lesson006c.py

```
5   result = print(
7       "is a sword that does",
8       sword_damage,
9       "damage, weighs",
10      sword_weight,
11      "kg and requires level",
12      sword_level
13      )
14  print(result)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powersh

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006c.py
Big Blade is a sword that does 6 damage, weighs 2.1 kg and requires level 3
None
PS C:\Users\hazeldotzone\Python Code>
```

- But we can't save that by having a variable refer to it!

The Build-A-String Workshop

- Enter the f-string!
- AKA format-string

lesson005functional.py | lesson005three.py | lesson005rebind.py | lesson006c.py | exercise005c.py | lesson002.py | lesson003.py | lesson004.py

lesson006c.py

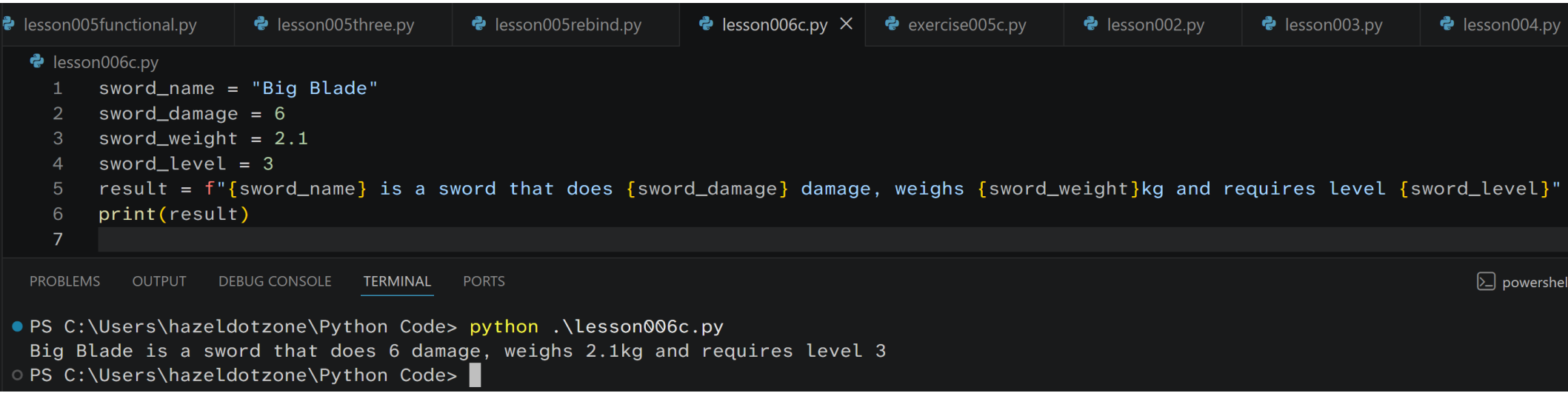
```
1  sword_name = "Big Blade"
2  sword_damage = 6
3  sword_weight = 2.1
4  sword_level = 3
5  result = f"{sword_name} is a sword that does {sword_damage} damage, weighs {sword_weight}kg and requires level {sword_level}"
6  print(result)
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershel

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006c.py
Big Blade is a sword that does 6 damage, weighs 2.1kg and requires level 3
PS C:\Users\hazeldotzone\Python Code>
```

The Build-A-String Workshop

- f-strings start with f and then a string delimiter

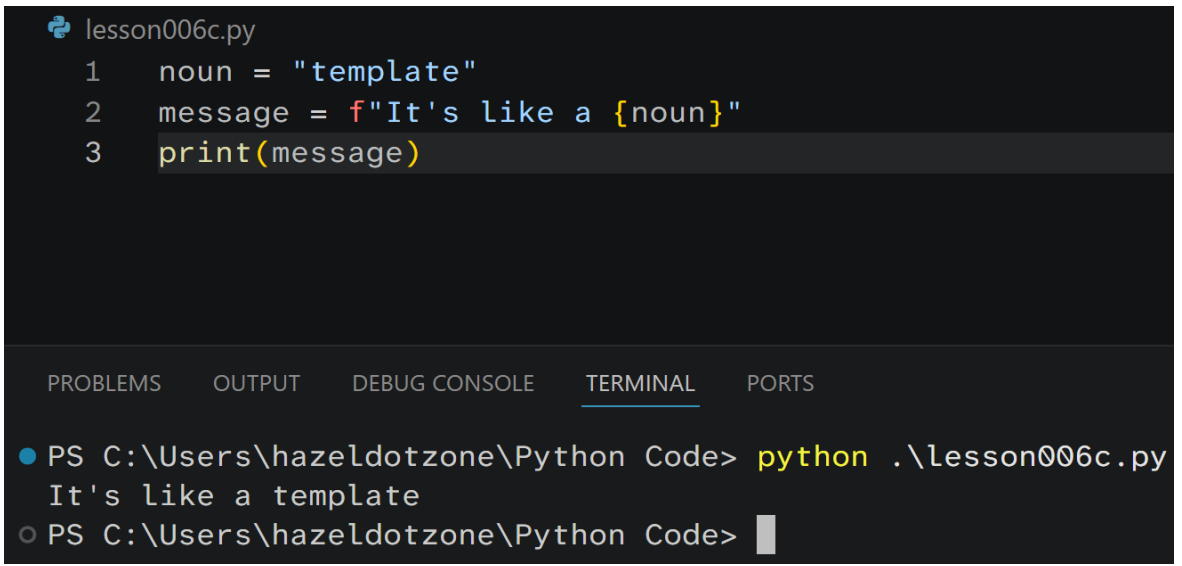


The Build-A-String Workshop

- f-strings start with f and then a string delimiter
- `f"stuff" f'stuff' f""stuff"" f'''stuff'''`
- We know `f"` can't be a number (it doesn't start with a number)
- We know `f"` can't be a name (names can't have " in them)

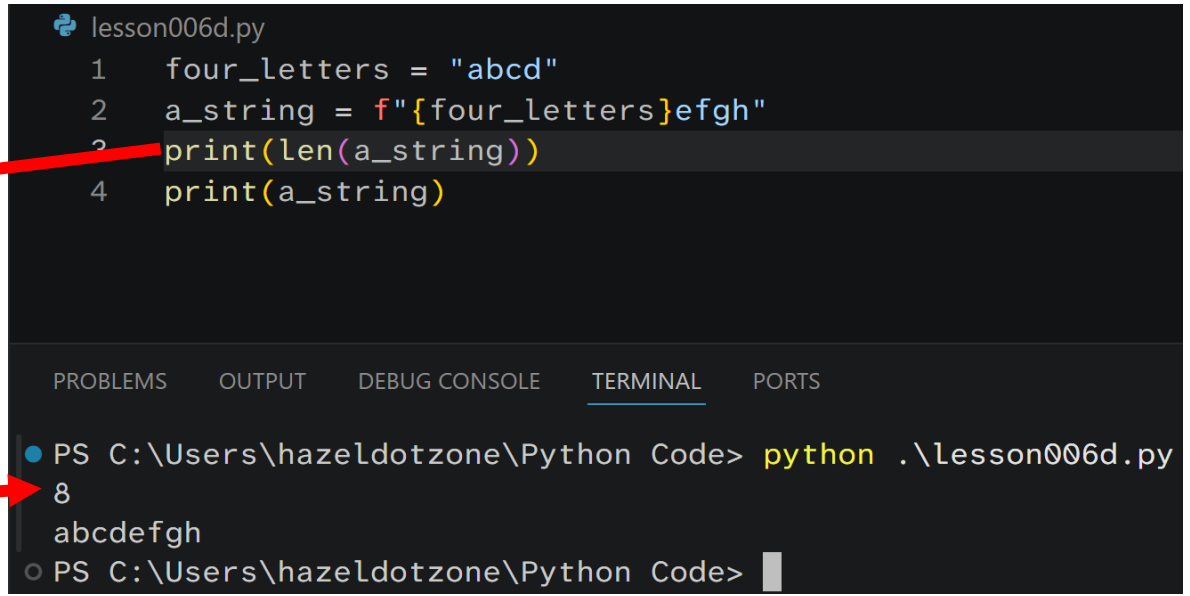
The Build-A-String Workshop

- It's like a template. That makes it easy to read and think about
  - Easy to read and think about → good software engineering!



The Build-A-String Workshop

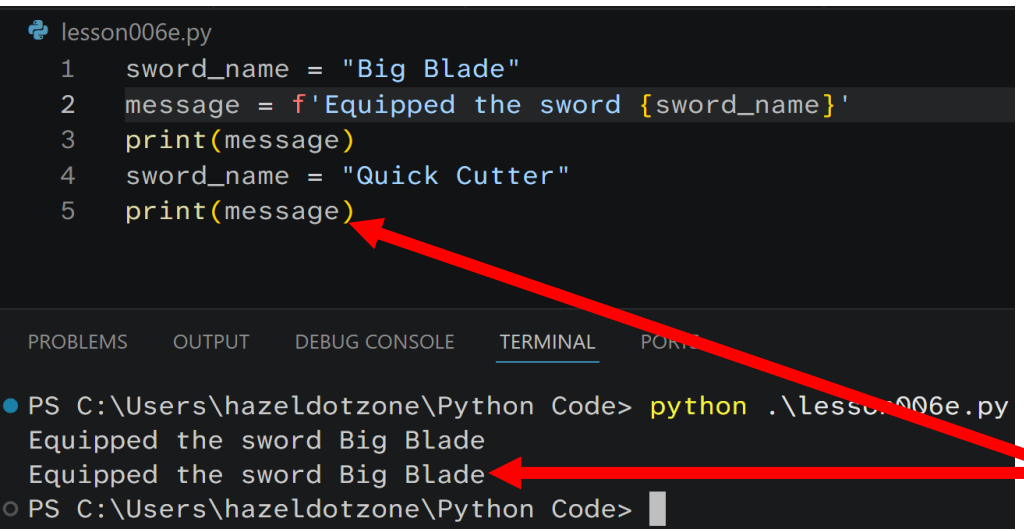
- The f-string is evaluated as soon as that line (statement) runs!



notice the length is 8 characters, not 18

The Build-A-String Workshop

- The f-string is evaluated as soon as that line (statement) runs!
- This means we can't save the template...



message didn't change

The Build-A-String Workshop

- The f-string is evaluated as soon as that line (statement) runs!
- This means we can't save the template...
- ... or can we???

The Build-A-String Workshop

- We can make a template as a normal string and save it by assigning a variable refer to it.

```
1  sword_name = "Big Blade"
2  template = 'Equipped the sword {sword_name}'
3  print(template)
4  sword_name = "Quick Cutter"
5  print(template)
```

normal string literal

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python .\lesson006f.py
Equipped the sword {sword_name}
Equipped the sword {sword_name}
PS C:\Users\hazeldotzone\Python Code>
```

The Build-A-String Workshop

- We can make a template as a normal string and save it by assigning a variable refer to it.

```
1  sword_name = "Big Blade"
2  template = 'Equipped the sword {sword_name}'
3  print(template)
4  sword_name = "Quick Cutter"
5  print(template)
```

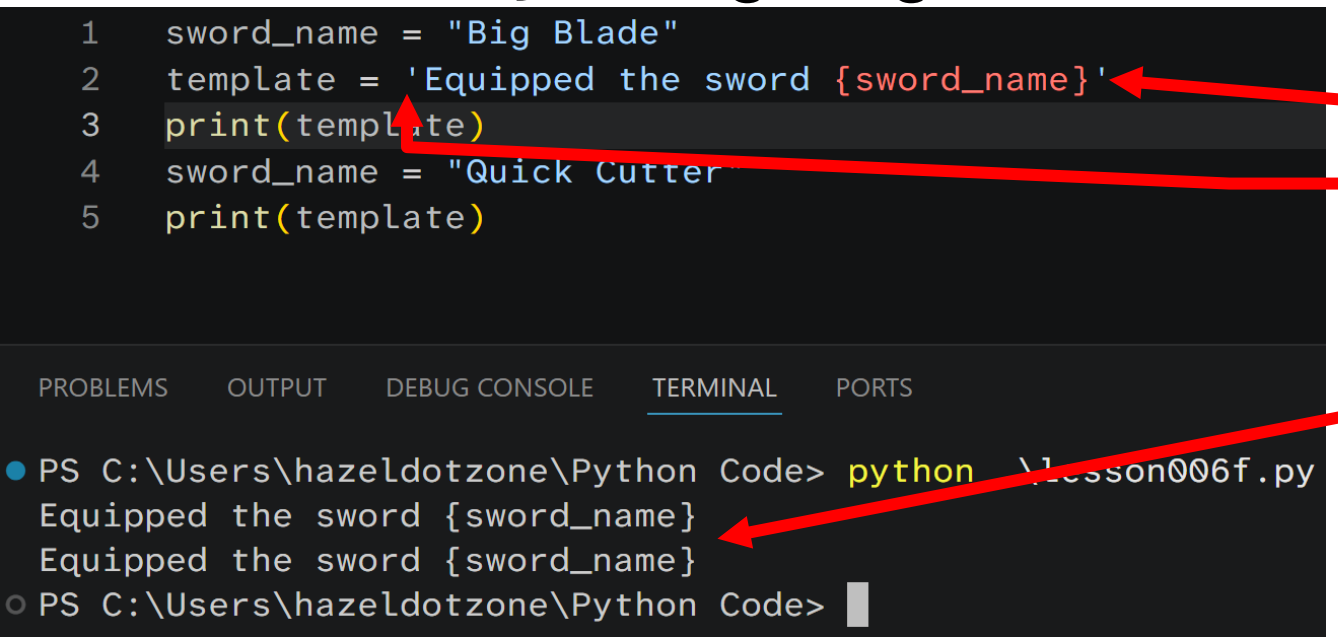
normal string literal
no f

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python .\lesson006f.py
Equipped the sword {sword_name}
Equipped the sword {sword_name}
PS C:\Users\hazeldotzone\Python Code>
```

The Build-A-String Workshop

- We can make a template as a normal string and save it by assigning a variable refer to it.

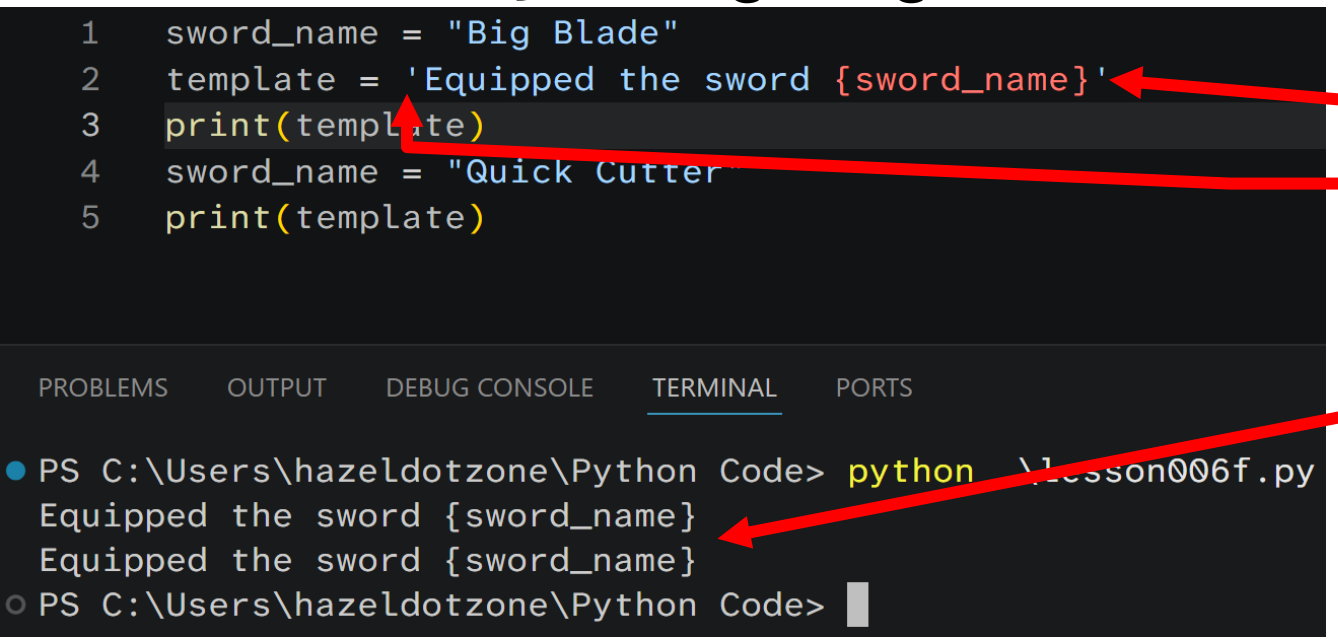


normal string literal
no f

prints the normal string literal... normally

The Build-A-String Workshop

- We can make a template as a normal string and save it by assigning a variable refer to it.



normal string literal
no f

prints the normal string literal... normally

The Build-A-String Workshop

- We can use the template by... calling the `.format` *method*

lesson006f.py

```
1   sword_name = "Big Blade"
2   template = 'Equipped the sword {sword_name}'
3   print(template.format(sword_name="Big Blade"))
4   print(template.format(sword_name="Quick Cutter"))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006f.py
Equipped the sword Big Blade
Equipped the sword Quick Cutter
PS C:\Users\hazeldotzone\Python Code>
```

Method Calling

```
print(template.format(sword_name="Big Blade"))
```

name

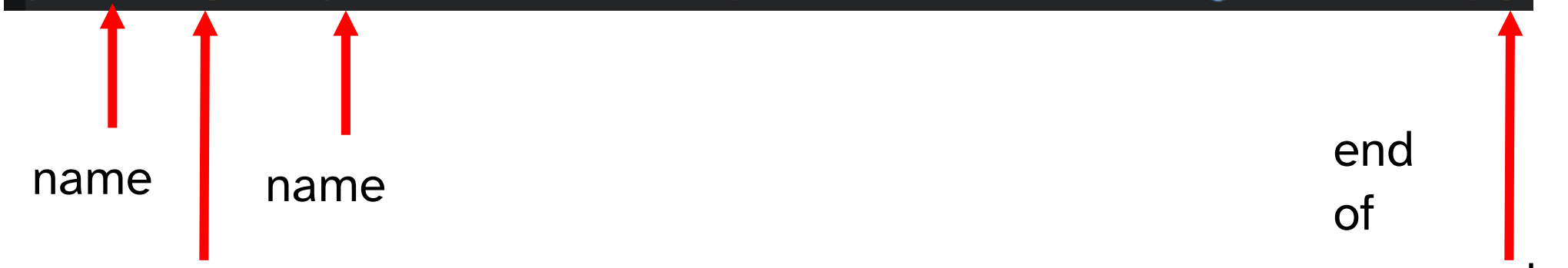
```
>>> print
<built-in function print>
```

Method Calling

```
print(template.format(sword_name="Big Blade"))
```

Method Calling

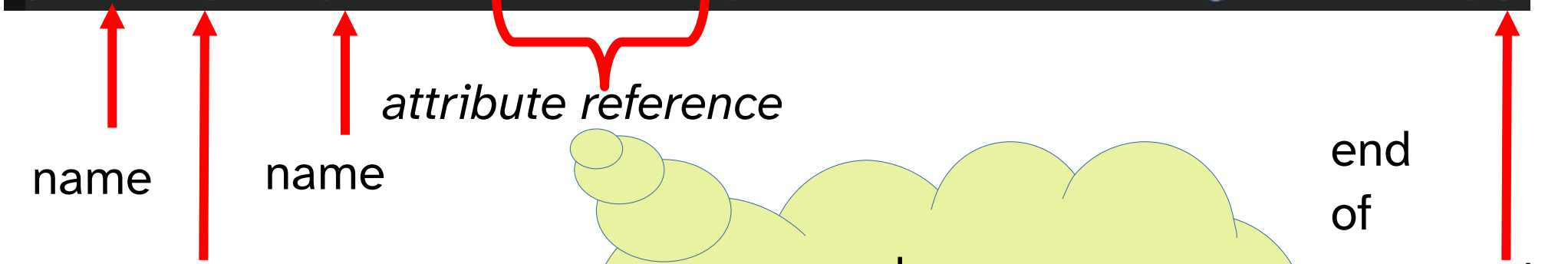
```
print(template.format(sword_name="Big Blade"))
```




```
>>> print(template)
Equipped the sword {sword_name}
```

Method Calling

```
print(template.format(sword_name="Big Blade"))
```



a . b

the . acts like English possesive form:

a's b

possessive form or

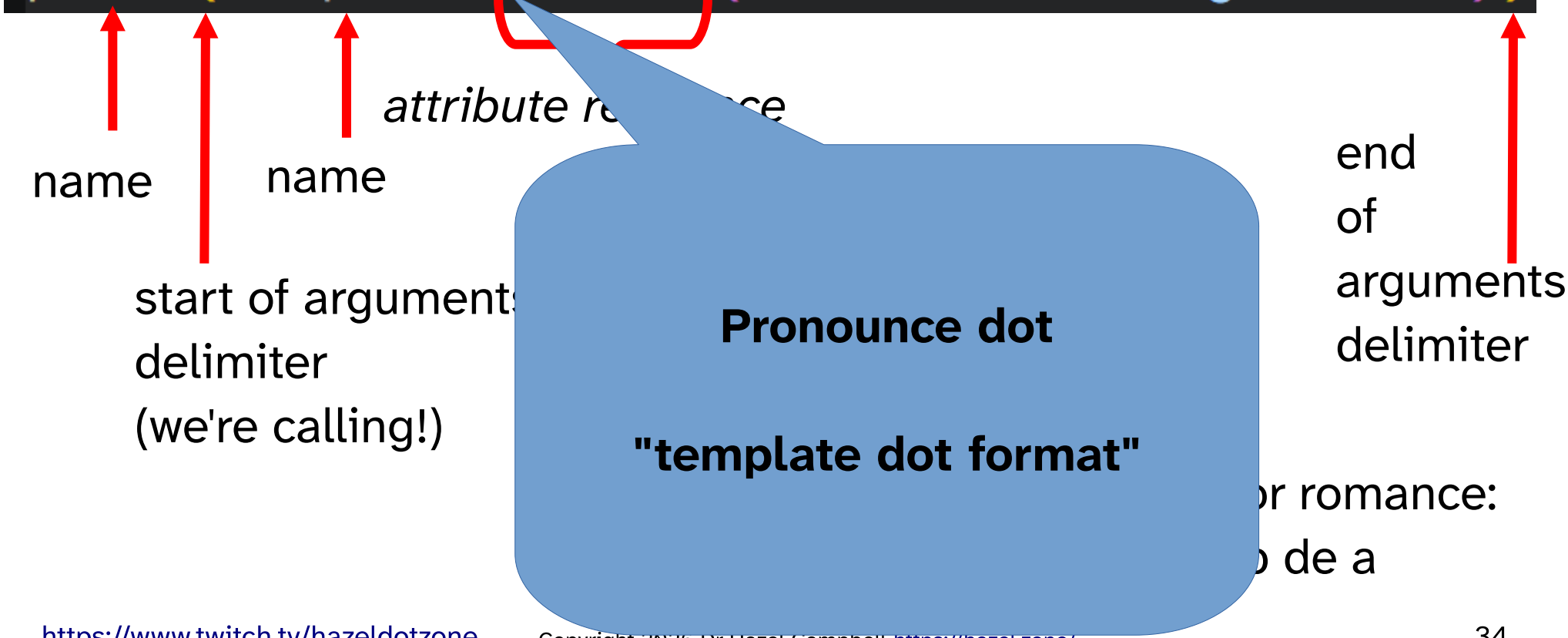
b of a

french:

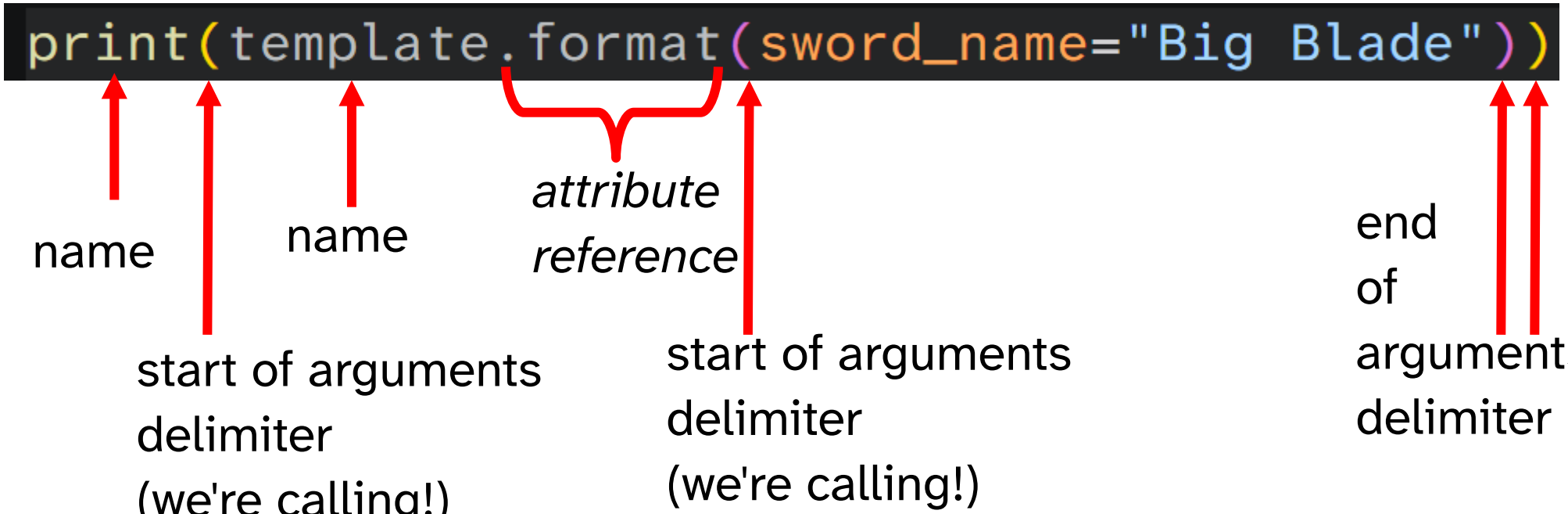
b de a

Method Calling

```
print(template.format(sword_name="Big Blade"))
```



Method Calling



Method Calling

```
print(template.format(sword_name="Big Blade"))
```

argument

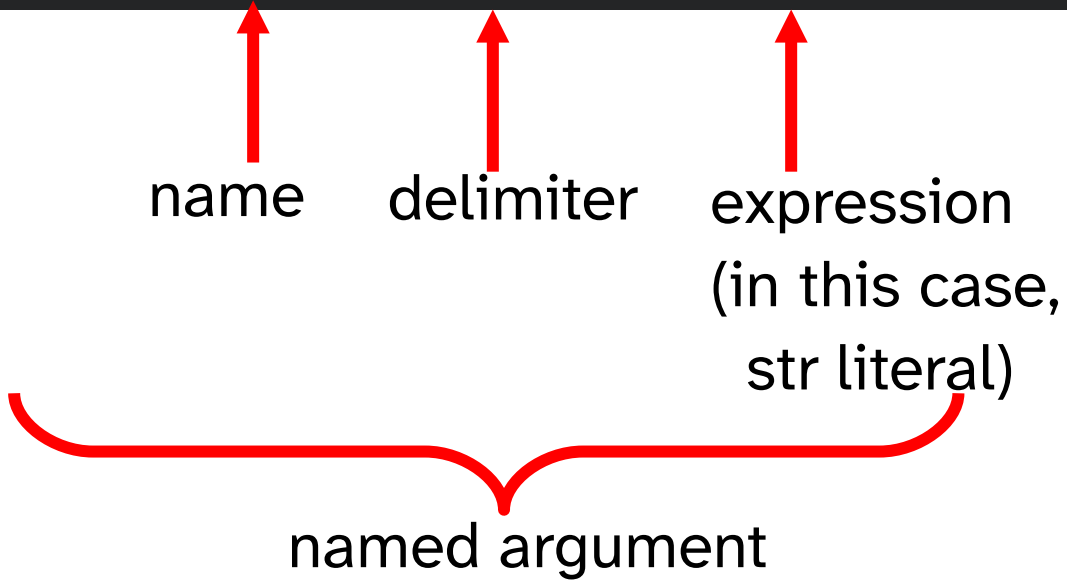
Method Calling

```
print(template.format(sword_name="Big Blade"))
```

named argument

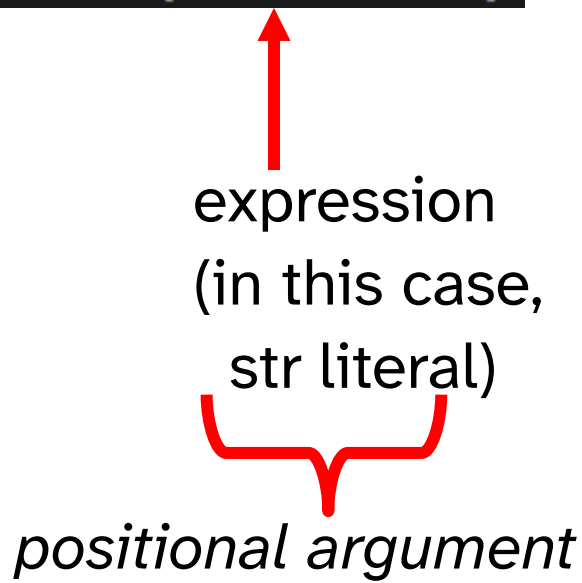
Named Arguments

```
print(template.format(sword_name="Big Blade"))
```

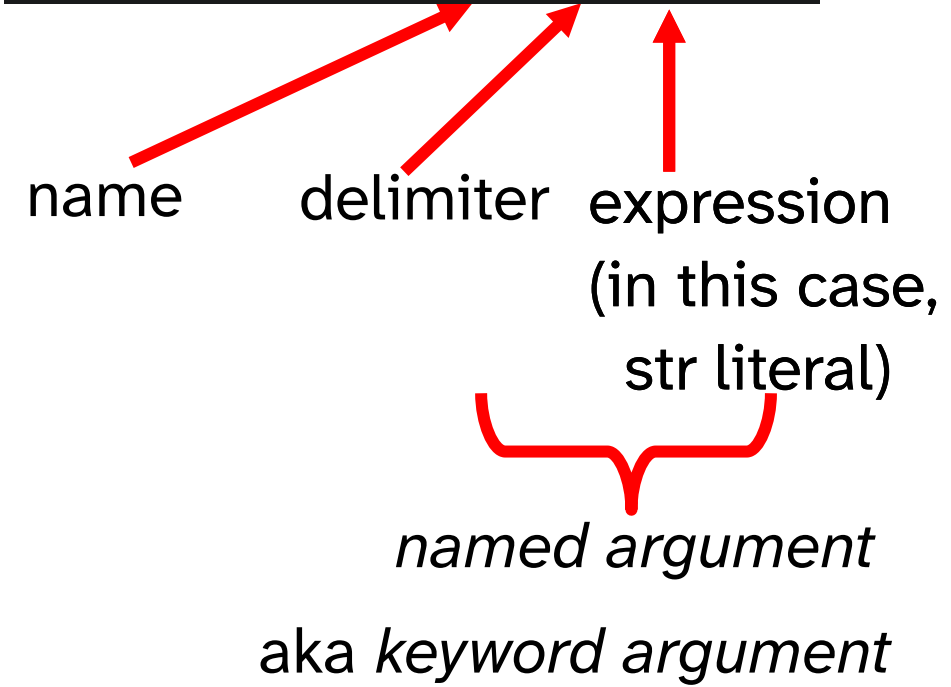


Named Arguments

```
>>> print("hello")
```




```
>>> print(end='\n')
```



Named Arguments

```
>>> print("hello")
```

expression
(in this case,
str literal)

positional argument

the thing we're printing

```
>>> print(end='\n')
```

name

delimiter

expression
(in this case,
str literal)

named argument

specify the line ending

Named Arguments

```
>>> print("hello")
```

expression
(in this case,
str literal)

positional argument

the thing we're printing

```
>>> print(end='\n')
```

name

delimiter

expression
(in this case,
str literal)

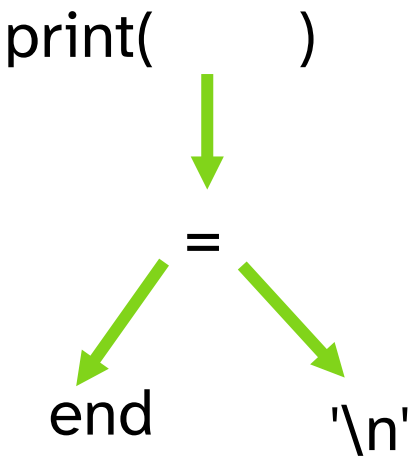
named argument

specify the line ending

Named Arguments

Syntax Diagram

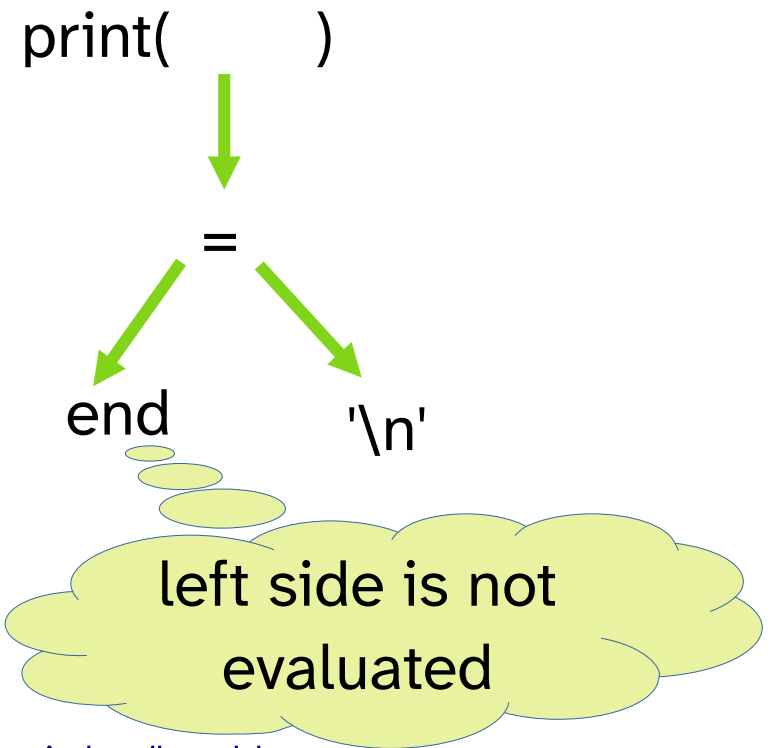
```
>>> print(end='\n')
```



Named Arguments

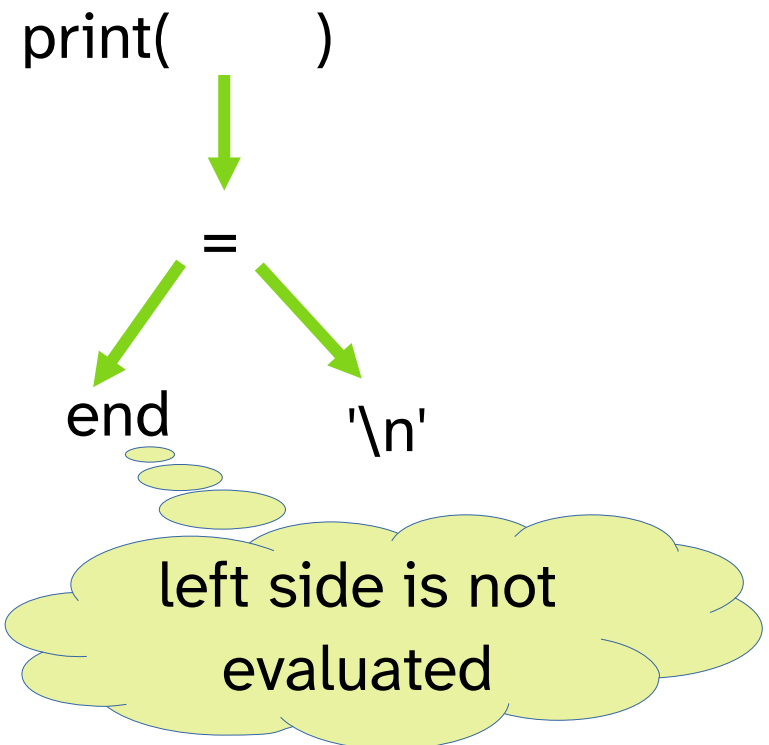
Syntax Diagram

```
>>> print(end='\n')
```

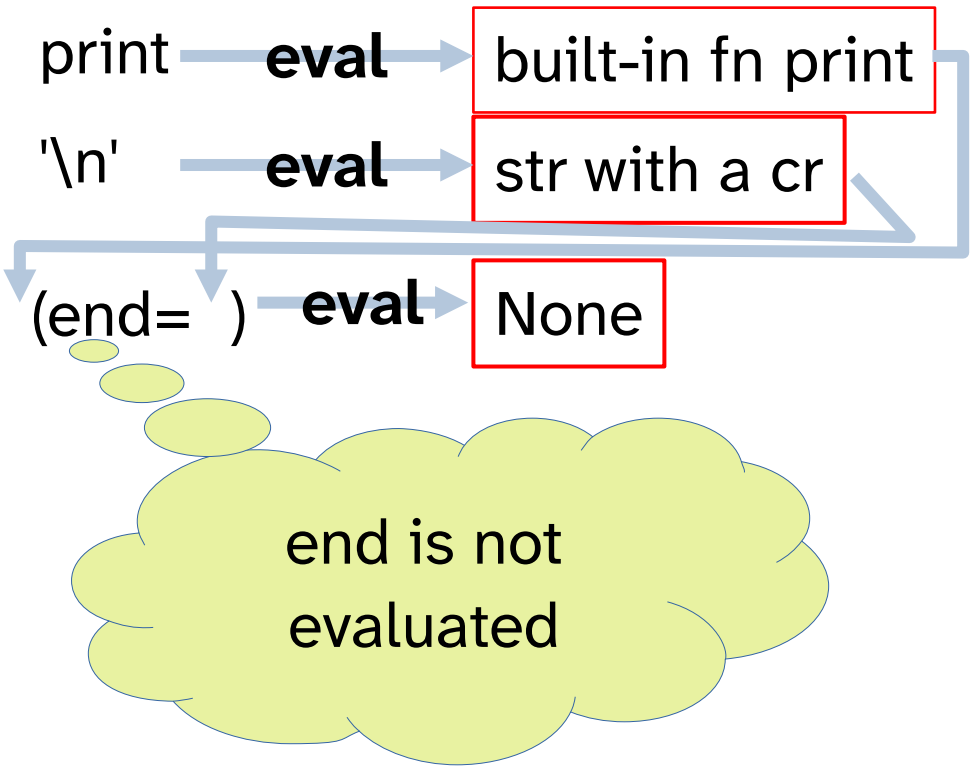


Named Arguments

Syntax Diagram

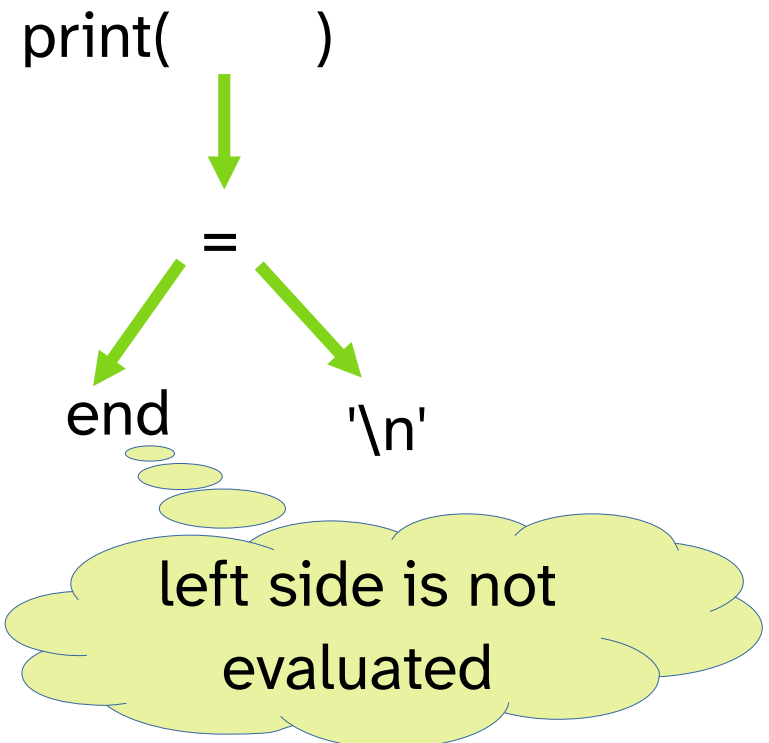



```
>>> print(end='\n')
```



Named Arguments

Syntax Diagram




```
>>> print(end='\n')
```

print **eval** → built-in fn print

'\n' **eval** → str with a cr

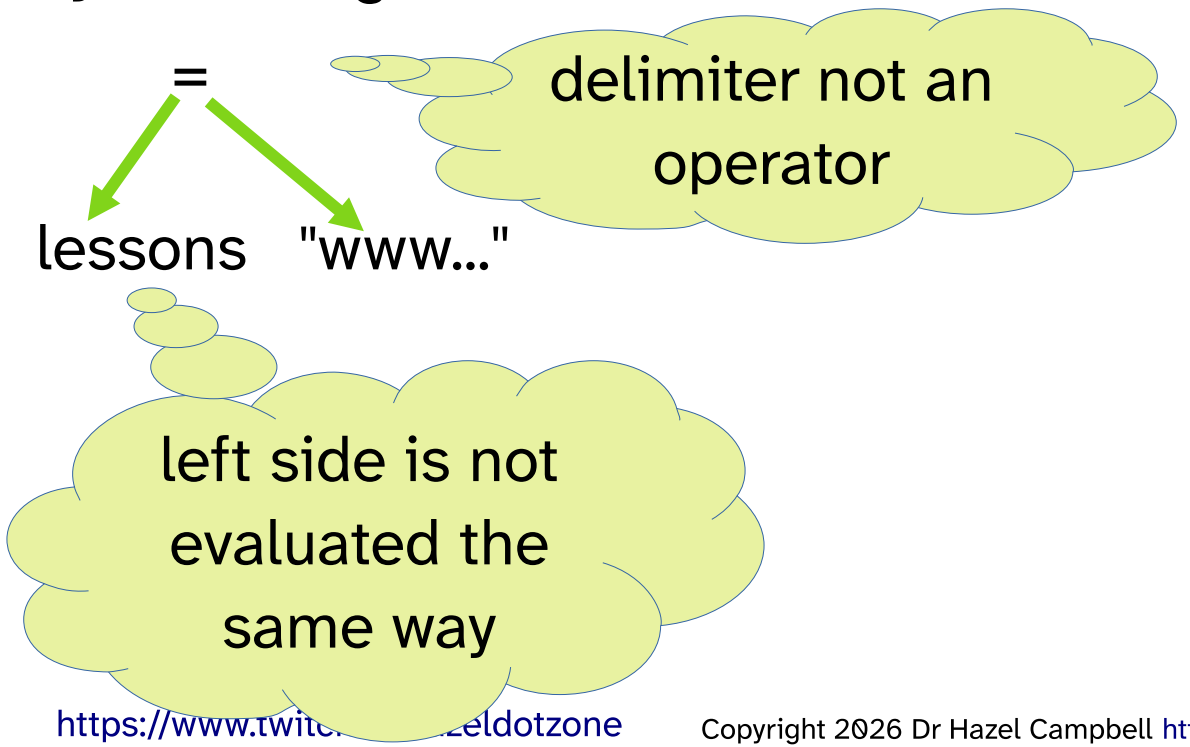
(end=) **eval** → None

the thing end refers to is the result of an evaluation

Remember: Assignment Statements

```
lessons="www.twitch.tv/hazeldotzone"
```

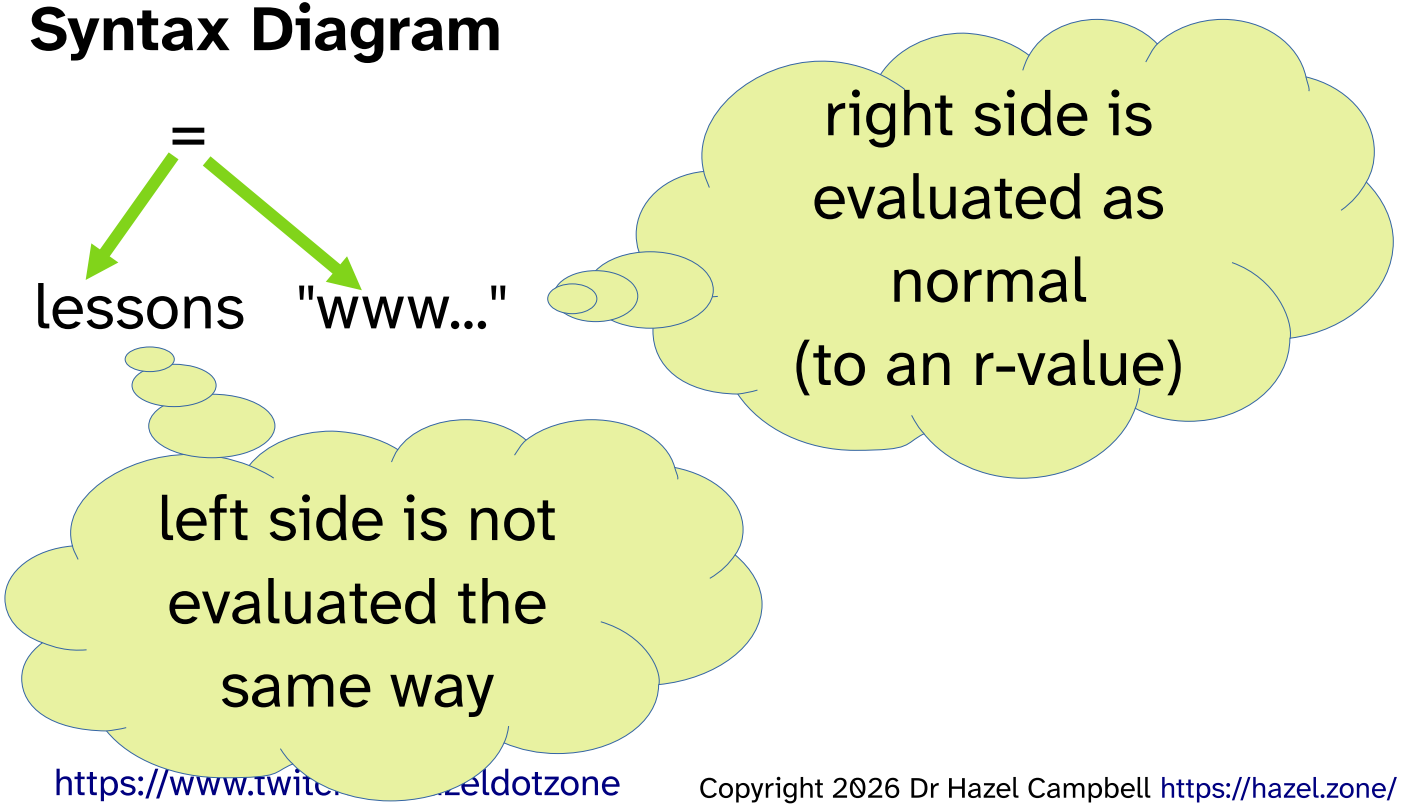
Syntax Diagram



Remember: Assignment Statements

```
lessons="www.twitch.tv/hazeldotzone"
```

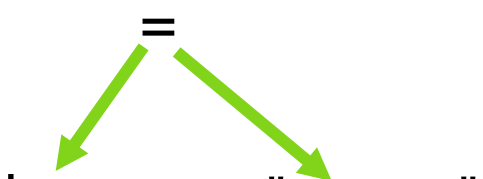
Syntax Diagram



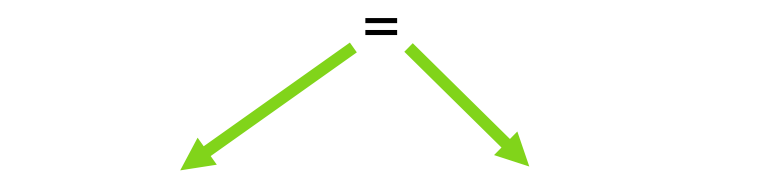
Remember: Assignment Statements

```
lessons="www.twitch.tv/hazeldotzone"
```

Syntax Diagram



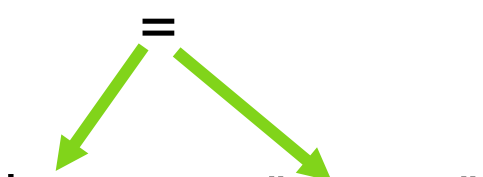
in general



Remember: Assignment Statements

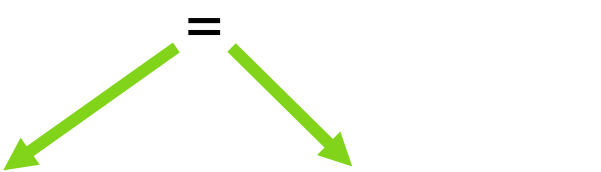
```
lessons="www.twitch.tv/hazeldotzone"
```

Syntax Diagram



in this case, the l-expression is only evaluated to see if we already have a `lesson` or if we need to create

in general



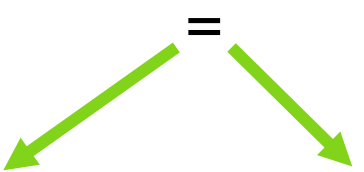
Remember: Assignment Statements

```
lessons="www.twitch.tv/hazeldotzone"
```

Syntax Diagram

in this case, the l-expression is only evaluated to see if we already have a `lesson` or if we need to create

in general



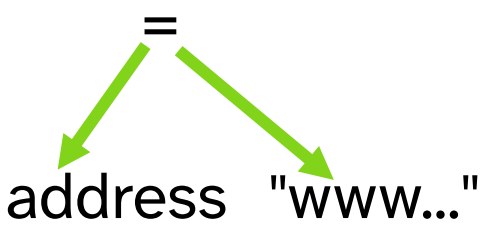
l-expression that will be evaluated to an l-value

r-expression that will be evaluated to an r-value

Remember: Assignment Statements

```
1   address="www.twitch.tv/hazeldotzone"
2   lessons=address
```

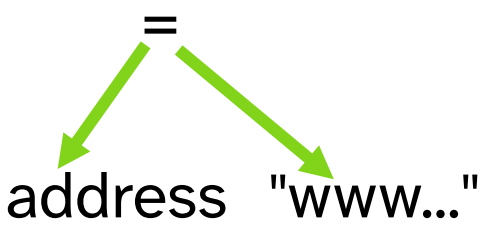
Line 1 Syntax Diagram

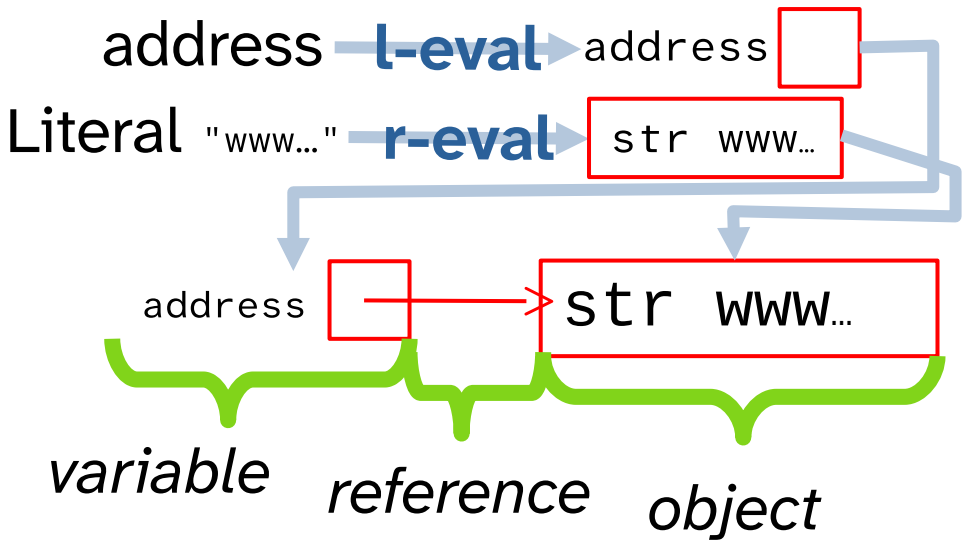


Remember: Assignment Statements

```
1   address="www.twitch.tv/hazeldotzone"
2   lessons=address
```

Line 1 Syntax Diagram

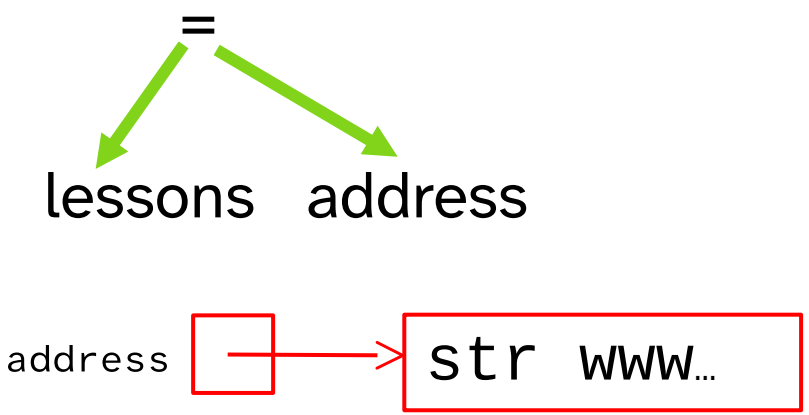


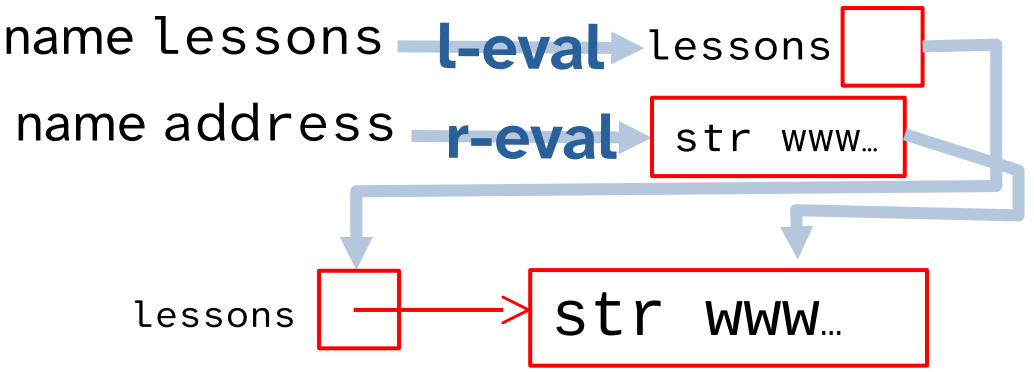


Remember: Assignment Statements

```
1   address="www.twitch.tv/hazeldotzone"
2   lessons=address
```

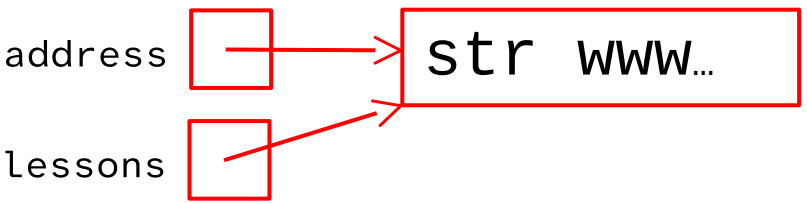
Line 2 Syntax Diagram





Remember: Assignment Statements

```
1   address="www.twitch.tv/hazeldotzone"
2   lessons=address
```



Named Arguments

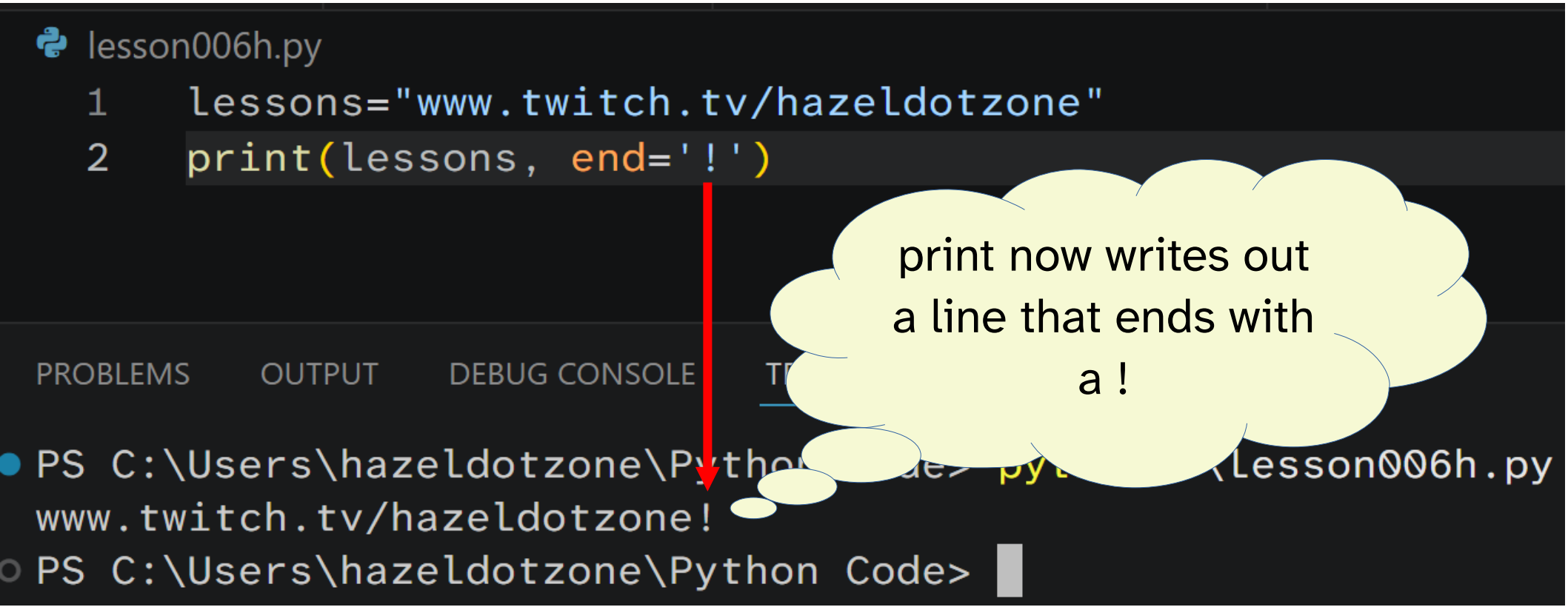
lesson006h.py

```
1   lessons="www.twitch.tv/hazeldotzone"
2   print(lessons, end='!')
```

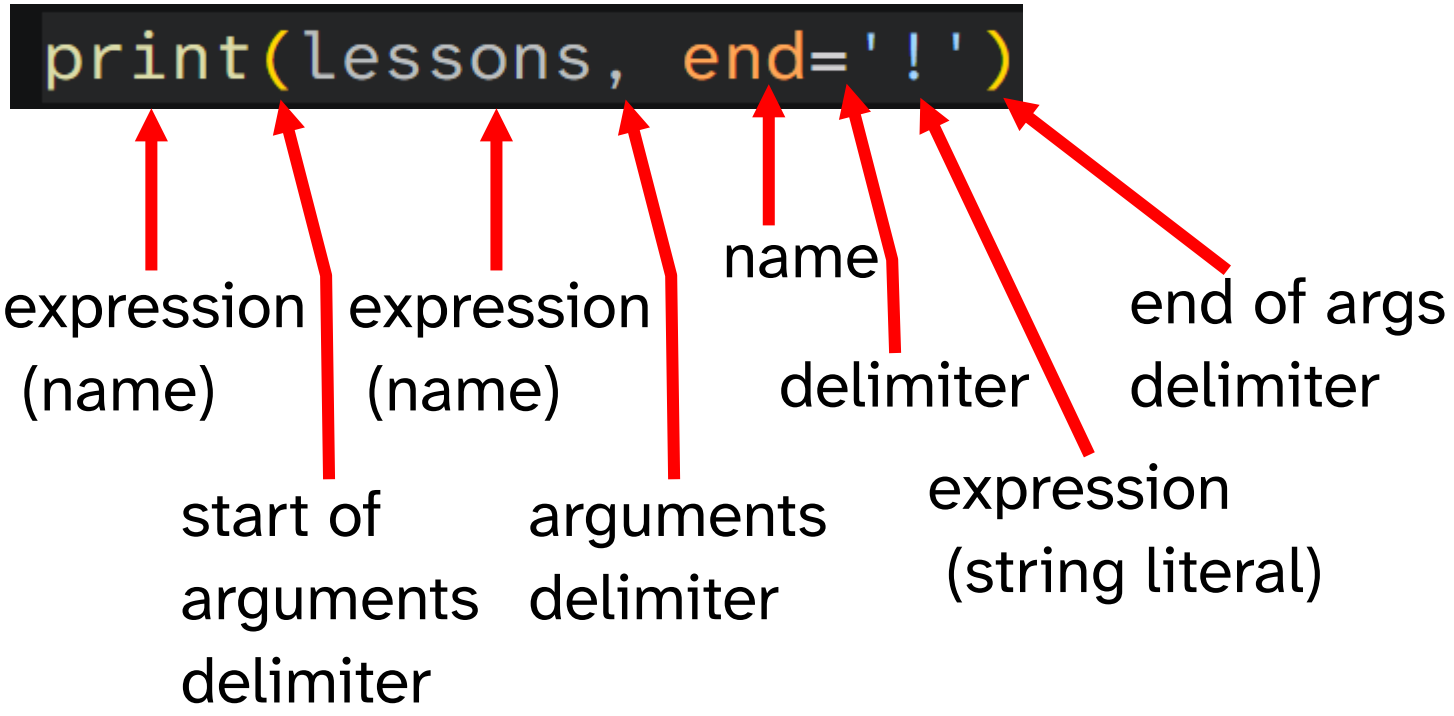
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006h.py
www.twitch.tv/hazeldotzone!
PS C:\Users\hazeldotzone\Python Code>
```

Named Arguments

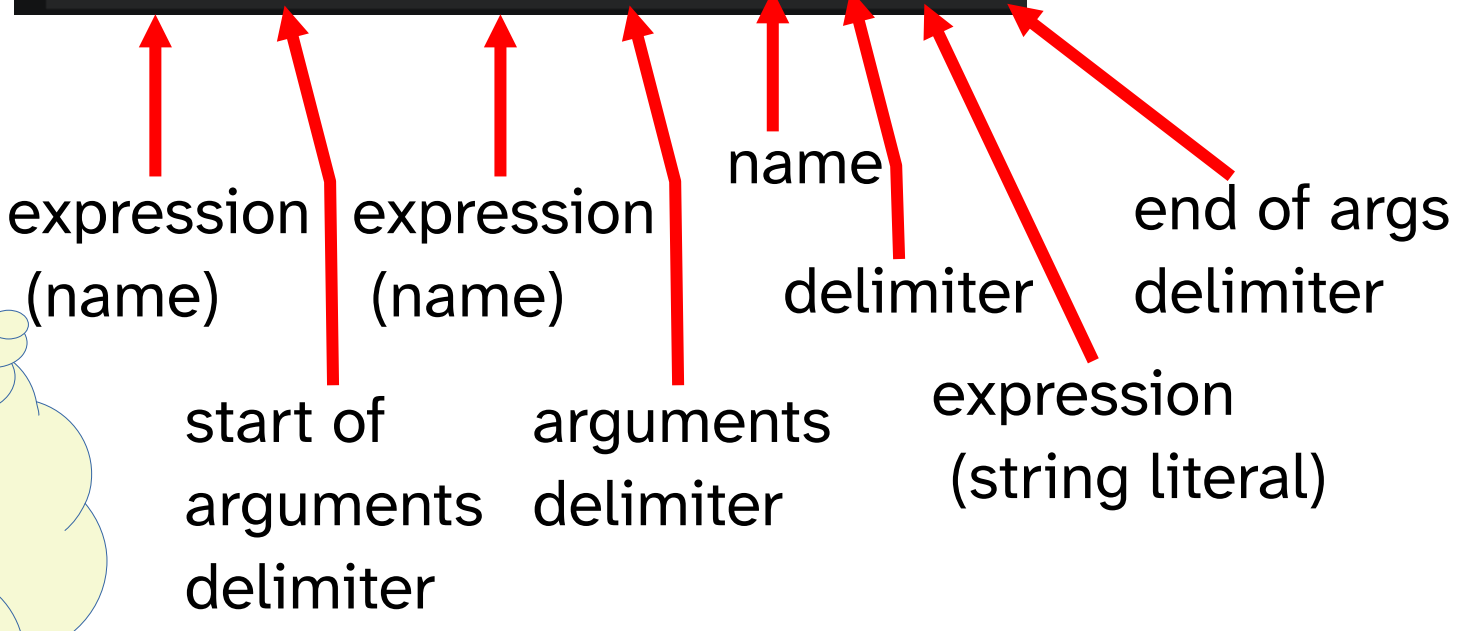


Named Arguments



Named Arguments

```
print(lessons, end='!')
```



is it really an expression?

Named Arguments

```
print(lessons, end='!')
```

expression (name)

is it really an expression?

```
1  lessons="www.twitch.tv/hazeldotzone"
2  print(lessons, end='!')
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python .\lesson006h.py
www.twitch.tv/hazeldotzone!
```

lesson006i.py

```
1  lessons="www.twitch.tv/hazeldotzone"
2  (print)(lessons, end='!')
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python .\lesson006h.py
www.twitch.tv/hazeldotzone!
PS C:\Users\hazeldotzone\Python Code> python .\lesson006i.py
www.twitch.tv/hazeldotzone!
PS C:\Users\hazeldotzone\Python Code>
```

https://www.twitch.tv/hazeldotzone

l-expressions vs r-expressions

lesson006j.py

```
1   adjective="fun"
2   noun="cat"
3   compound_noun = adjective + " " + noun
4   print(compound_noun)
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006j.py
fun cat
PS C:\Users\hazeldotzone\Python Code>
```

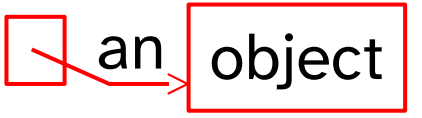
we can put operators like +
in an r-expression

in general

=

l-expression that will be evaluated to a reference

r-expression that will be evaluated to an object



l-expressions vs r-expressions

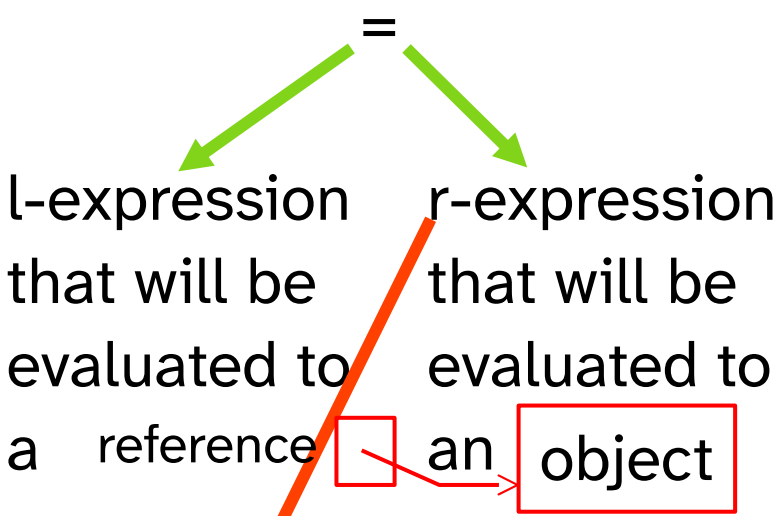
lesson006j.py

```
1  adjective="fun"
2  noun="cat"
3  compound_noun = adjective + " " + noun
4  print(compound_noun)
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006j.py
fun cat
PS C:\Users\hazeldotzone\Python Code>
```

in general



we can put operators like +
in an r-expression

which python just calls "expression"

l-expressions vs r-expressions

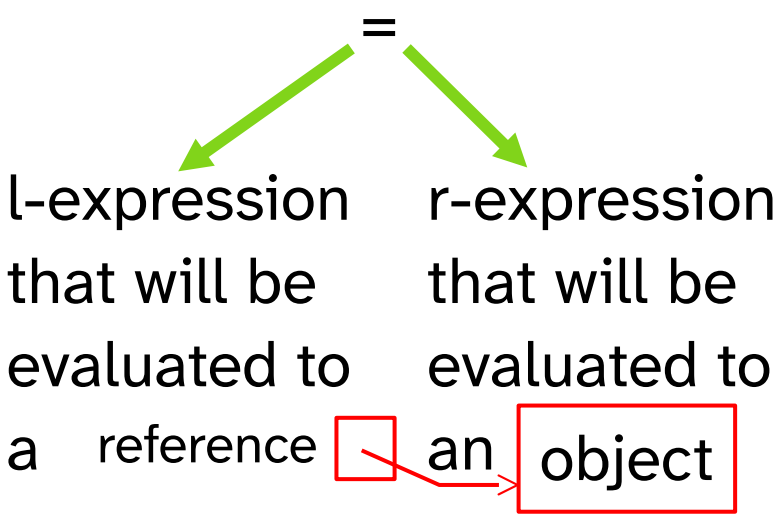
lesson006j.py

```
1  adjective="fun"
2  noun="cat"
3  a + b = noun
4  print(a)
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006j.py
  File "C:\Users\hazeldotzone\Python Code\lesson006j.py", line 3
    a + b = noun
    ^^^^^
SyntaxError: cannot assign to expression here. Maybe you meant '==' instead of '='?
PS C:\Users\hazeldotzone\Python Code>
```

in general



we can't put operators like + in an l-expression

l-expressions vs r-expressions

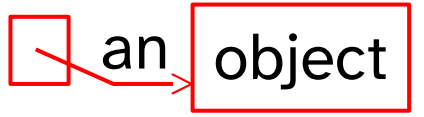
```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006j.py
  File "C:\Users\hazeldotzone\Python Code\lesson006j.py", line 3
    a == b = noun
    ^^^^^^
SyntaxError: cannot assign to comparison
PS C:\Users\hazeldotzone\Python Code>
```

in general

=

l-expression that will be evaluated to a reference

r-expression that will be evaluated to an object



we can't put operators like == in an l-expression

which python calls "target"

l-expressions vs r-expressions

lesson006j.py

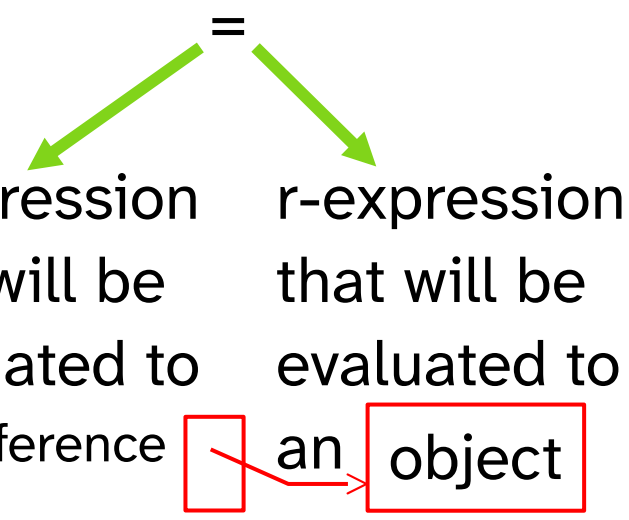
```
1   adjective="fun"
2   noun="cat"
3   (compound_noun) = adjective + " " + noun
4   print(compound_noun)
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006j.py
fun cat
PS C:\Users\hazeldotzone\Python Code>
```

we CAN use some things like ( ) in an l-expression

in general



l-expressions vs r-expressions

lesson006j.py

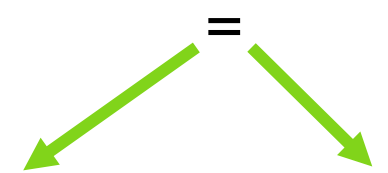
```
1   adjective="fun"
2   noun="cat"
3   (compound_noun) = adjective + " " + noun
4   print(compound_noun)
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006j.py
fun cat
PS C:\Users\hazeldotzone\Python Code>
```

we CAN use some things like ( ) in an l-expression sometimes

in general



l-expression that will be evaluated to a reference

r-expression that will be evaluated to an object

l-expressions vs r-expressions

lesson006j.py

```
1   adjective="fun"
2   noun="cat"
3   (compound_noun) = adjective + " " + noun
4   print(compound_noun)
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006j.py
fun cat
PS C:\Users\hazeldotzone\Python Code>
```

r-expressions evaluate to object

in general

=

l-expression that will be evaluated to a reference

r-expression that will be evaluated to an object

l-expressions vs r-expressions

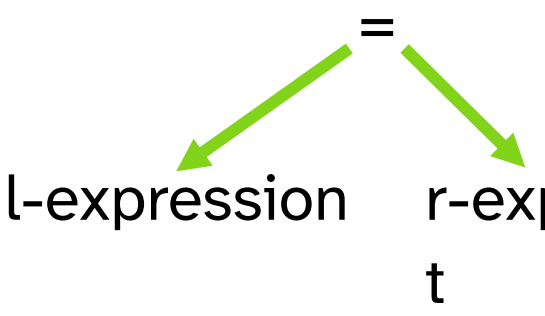
lesson006j.py

```
1   adjective="fun"
2   noun="cat"
3   (compound_noun) = adjective + " " + noun
4   print(compound_noun)
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006j.py
fun cat
PS C:\Users\hazeldotzone\Python Code>
```

in general



l-expressions evaluate to reference **that will refer to an** object

Left side: is it an expression?

- Can we put () around it? Yes!
- We will see other operations that work on the left and right side.

Is it an r-expression?

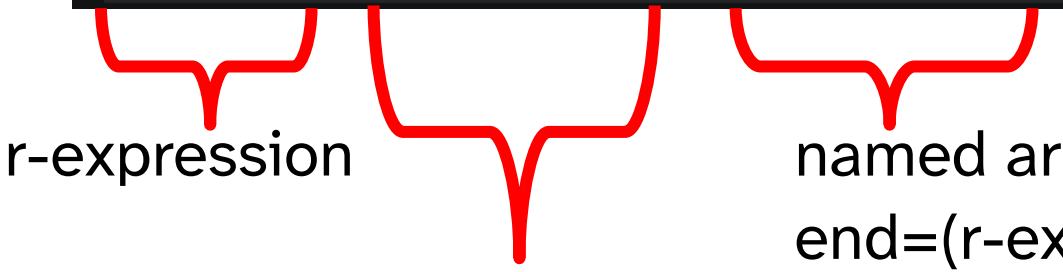
- Can we print it?
- Can we use it on the right side of an assignment statement so we can have a variable refer to it?
- Can we put it by itself on a line? (expression statement)

Is it an l-expression?

- Can we use it in an r-expression to get the object it refers to?
- Is it on the left side of an assignment statement?

Back to Named Arguments

```
print(lessons, end='!')
```



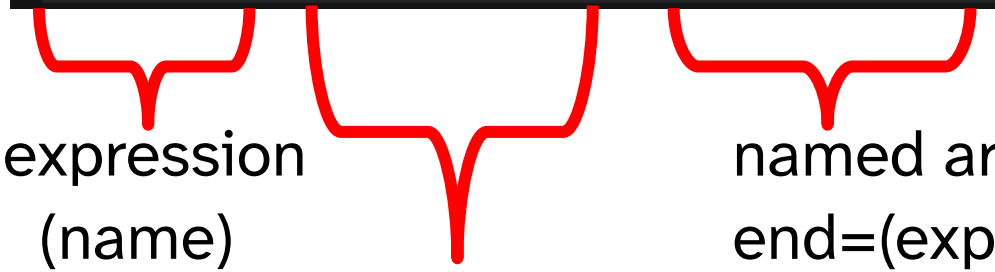
r-expression

positional argument (r-expression)

named argument end=(r-expression)

Named Arguments

```
print(lessons, end='!')
```



expression
(name)

positional argument
(expression)
(name)

named argument
end=(expression)
(str literal)

Named Arguments: is it an expression?

```
print(lessons, (end)='!')
```

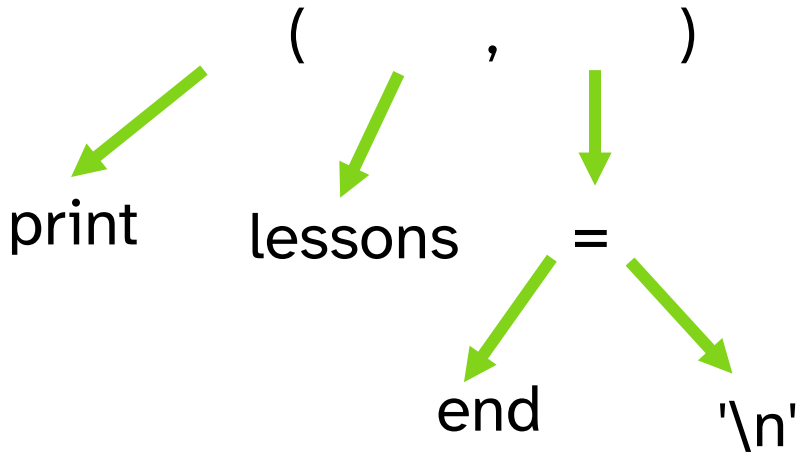
```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006i2.py
  File "C:\Users\hazeldotzone\Python Code\lesson006i2.py", line 3
    print(lessons, (end)='!')
                    ^^^^^
SyntaxError: expression cannot contain assignment, perhaps you meant "=="?
PS C:\Users\hazeldotzone\Python Code>
```

nope it's just a name

Named Arguments

Syntax Diagram

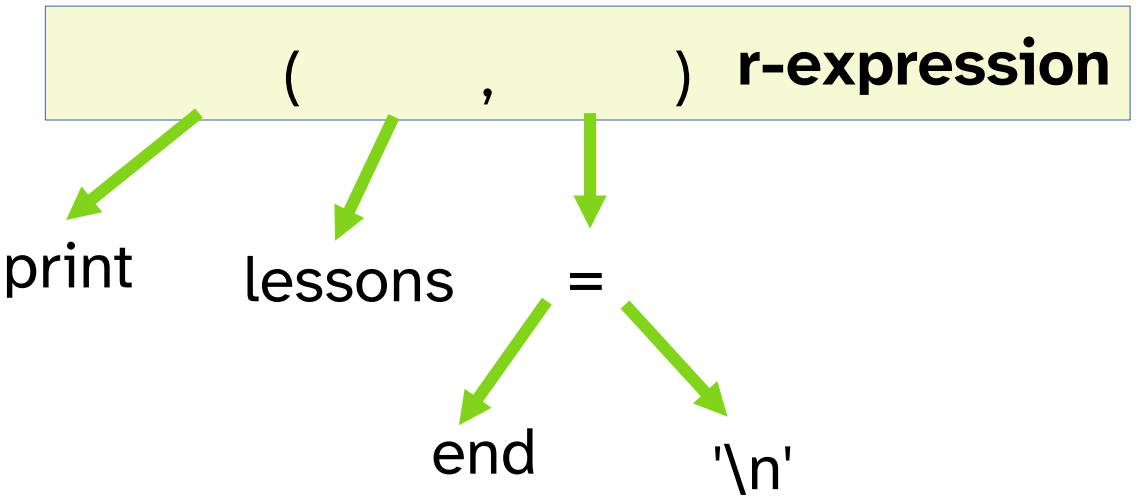
```
print(lessons, end='!')
```



Named Arguments

Syntax Diagram

```
print(lessons, end='!')
```

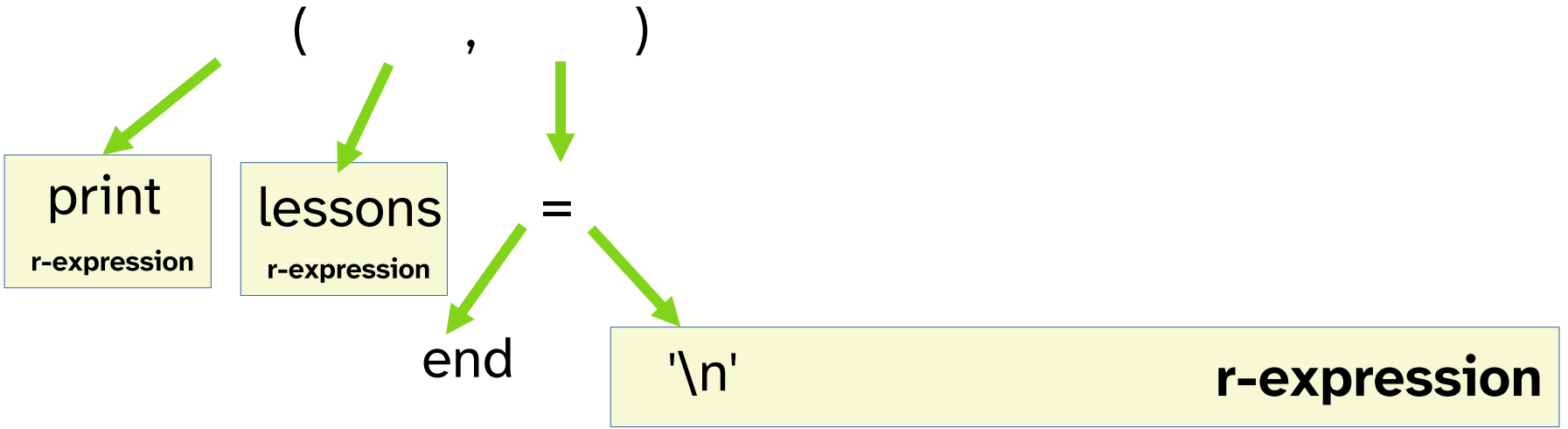


expression statements are just r-expressions in a statement by itself

Named Arguments

Syntax Diagram

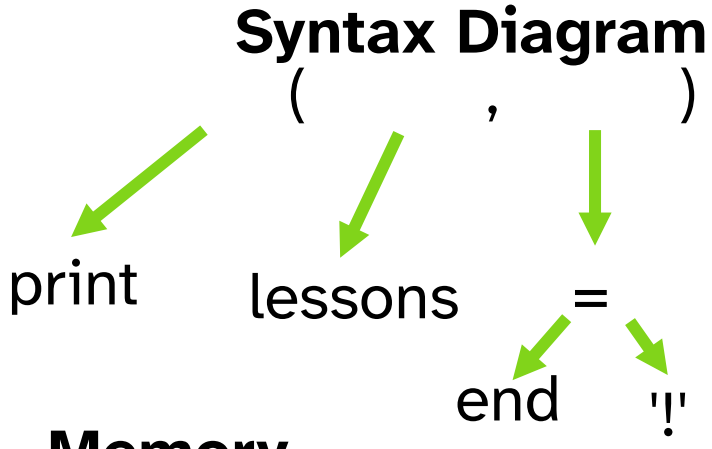
```
print(lessons, end='!')
```

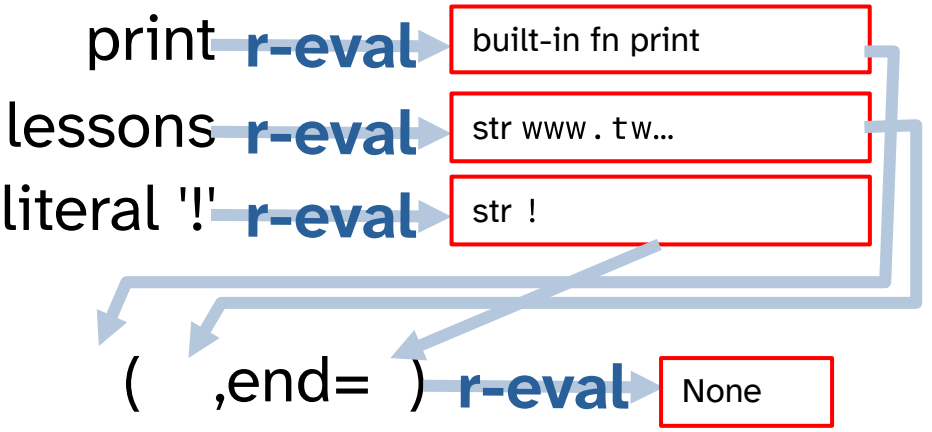


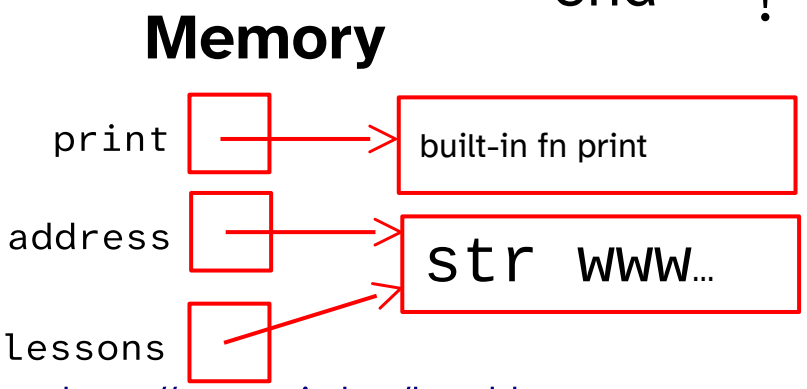

```
3   (print)((lessons), end=('!')) # works fine too
```

Named Arguments

```
print(lessons, end='!')
```



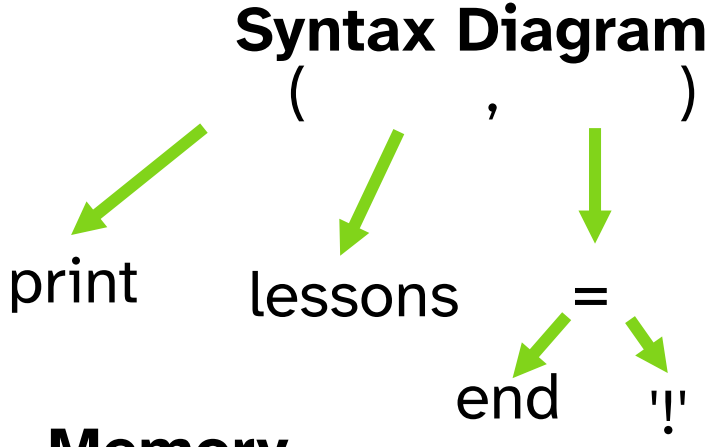


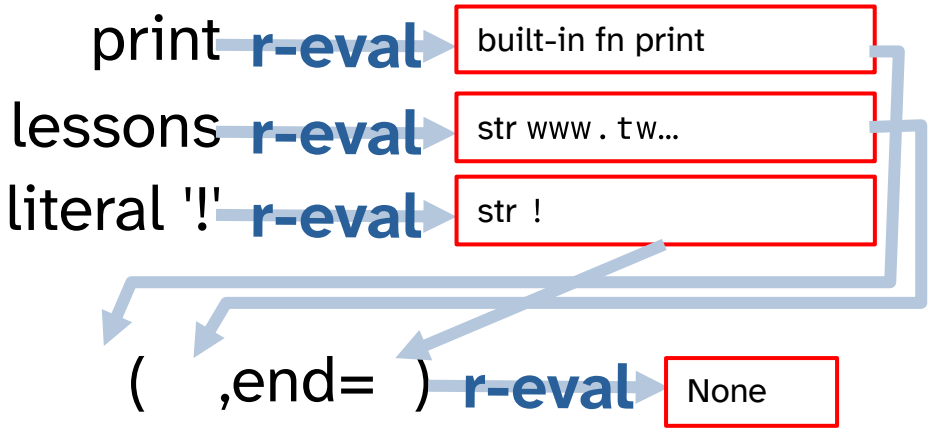


Named Arguments

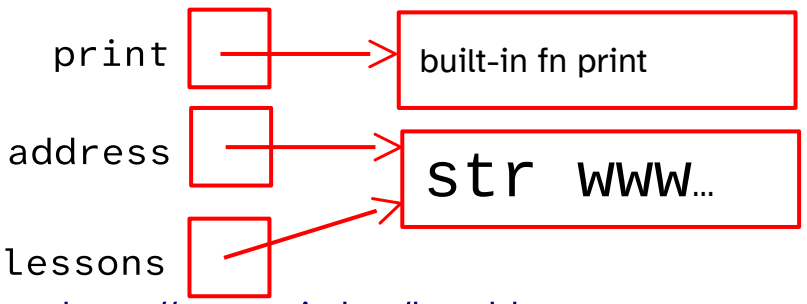
```
print(lessons, end='!')
```

Syntax Diagram



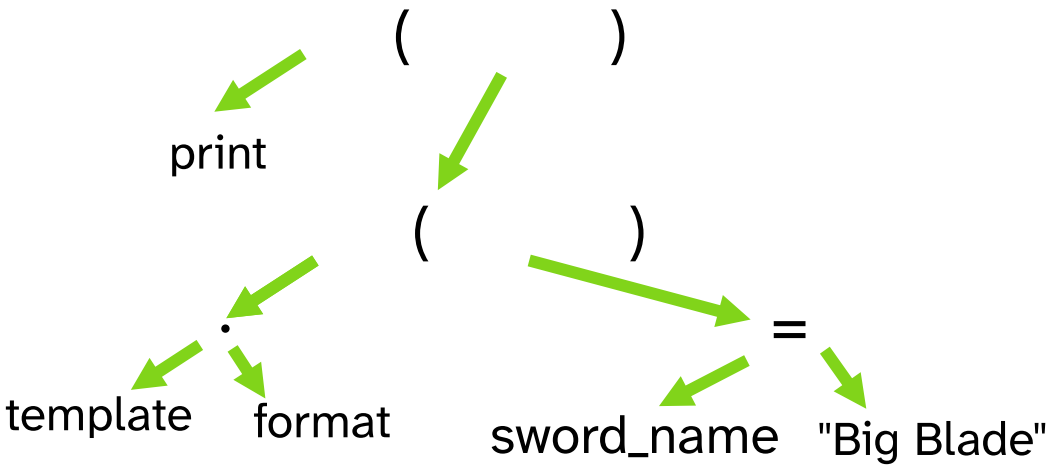


Memory



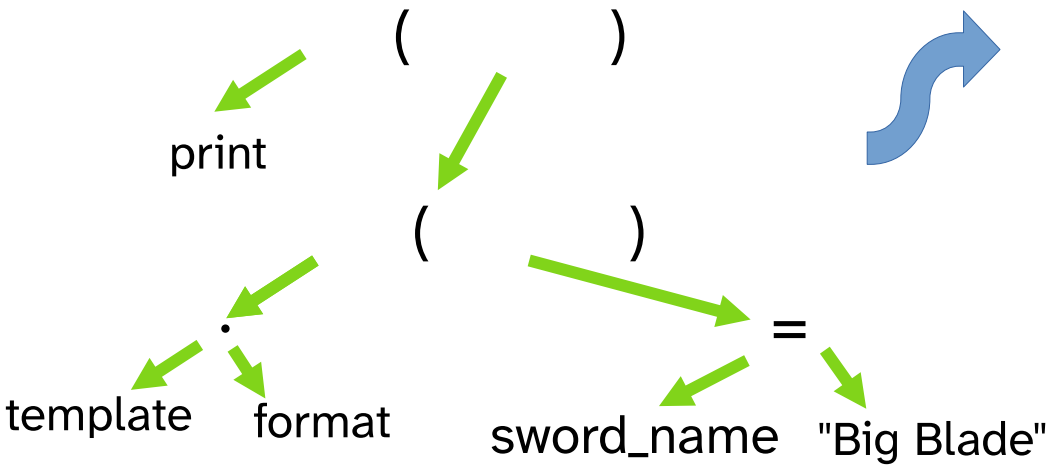
Back to this again

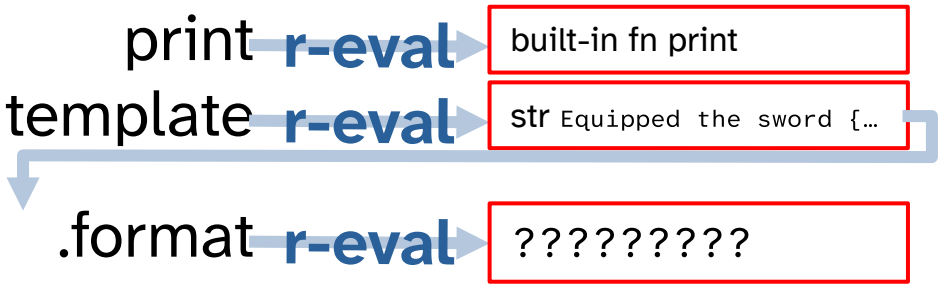
```
1    sword_name = "Big Blade"
2    template = 'Equipped the sword {sword_name}'
3    print(template.format(sword_name="Big Blade"))
```



Back to this again

```
1   sword_name = "Big Blade"
2   template = 'Equipped the sword {sword_name}'
3   print(template.format(sword_name="Big Blade"))
```





Back to this again

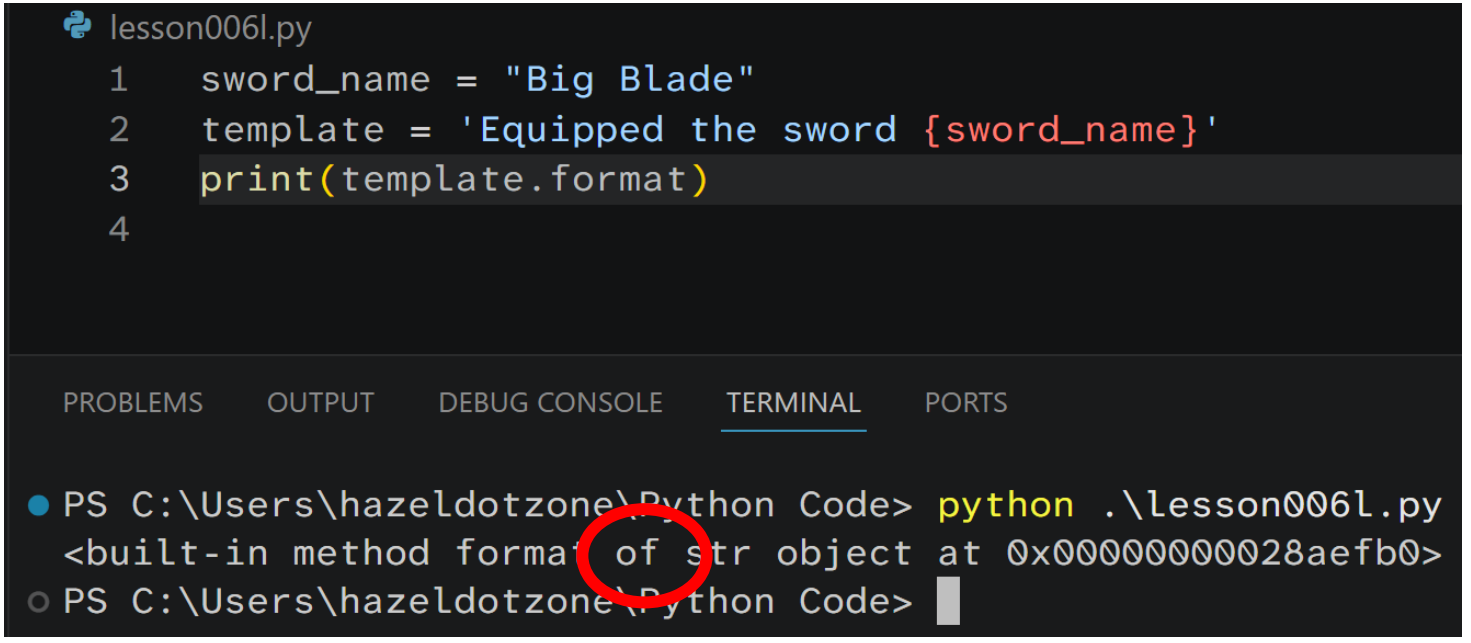
lesson006l.py

```
1   sword_name = "Big Blade"
2   template = 'Equipped the sword {sword_name}'
3   print(template.format)
4
```

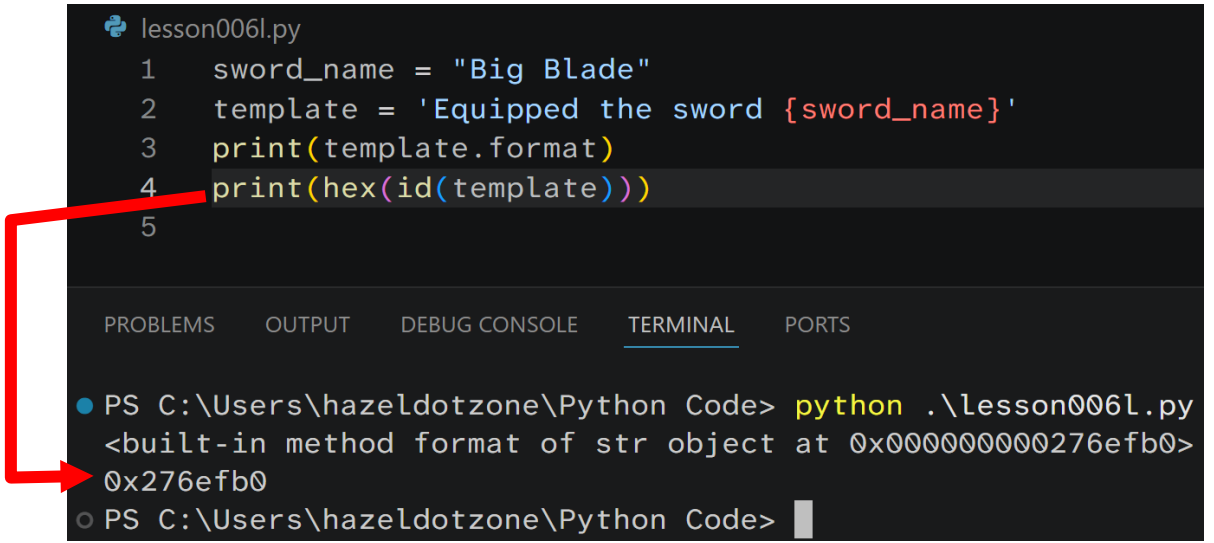
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson006l.py
<built-in method format of str object at 0x00000000028aefb0>
PS C:\Users\hazeldotzone\Python Code>
```

Back to this again

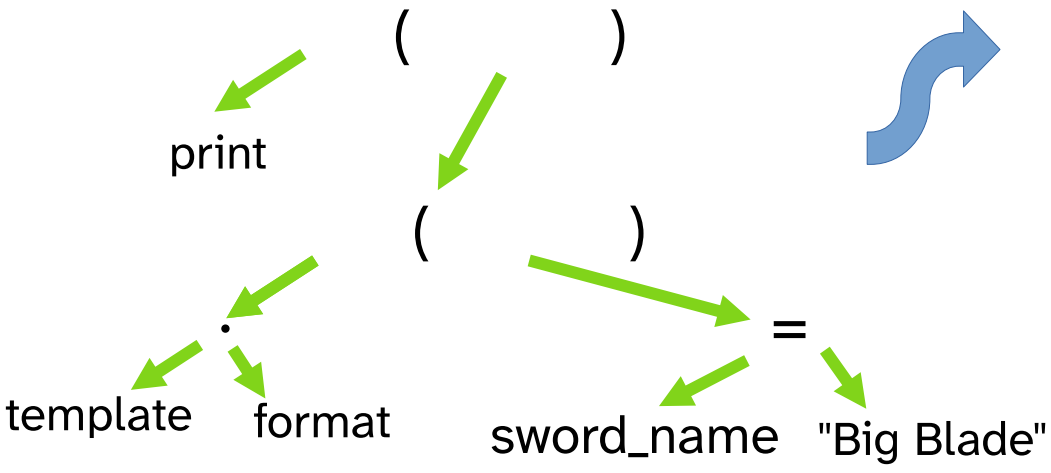


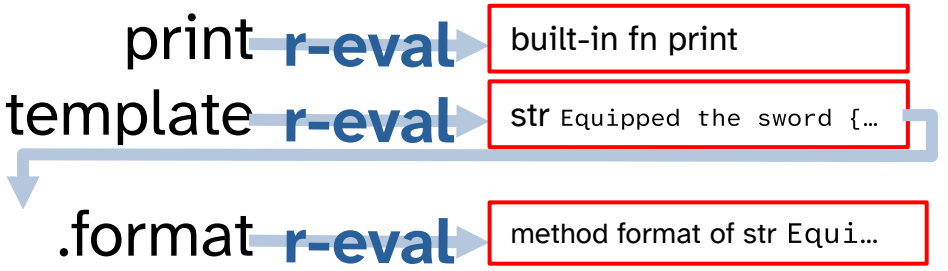
Back to this again



Back to this again

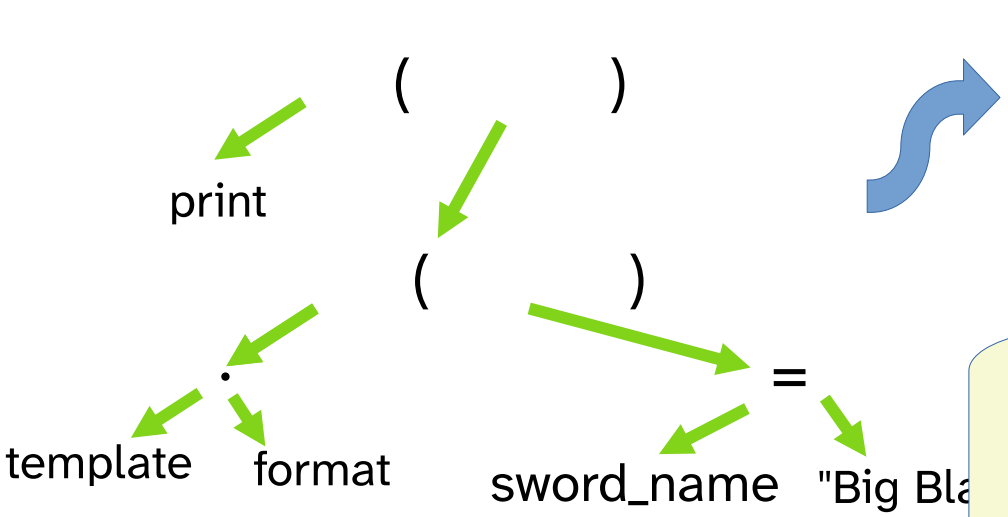
```
1    sword_name = "Big Blade"
2    template = 'Equipped the sword {sword_name}'
3    print(template.format(sword_name="Big Blade"))
```

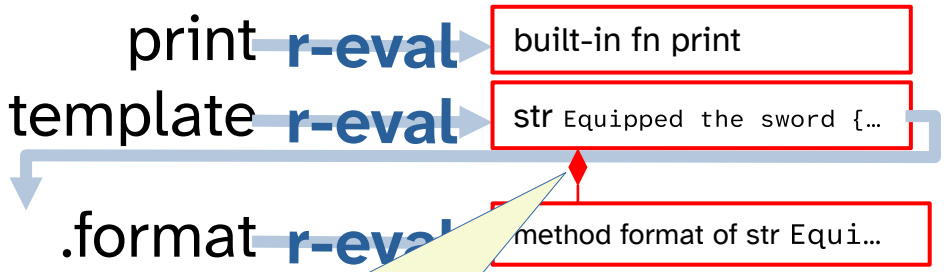




Back to this again

```
1   sword_name = "Big Blade"
2   template = 'Equipped the sword {sword_name}'
3   print(template.format(sword_name="Big Blade"))
```

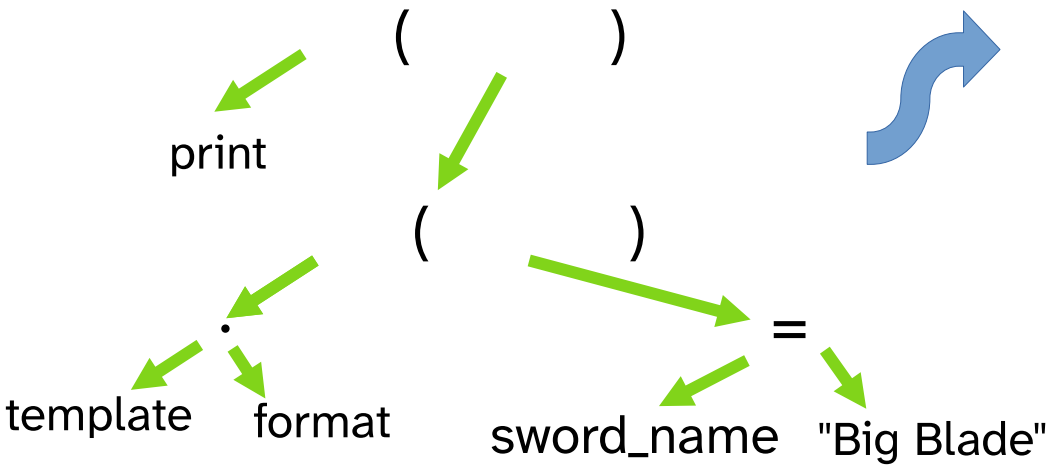


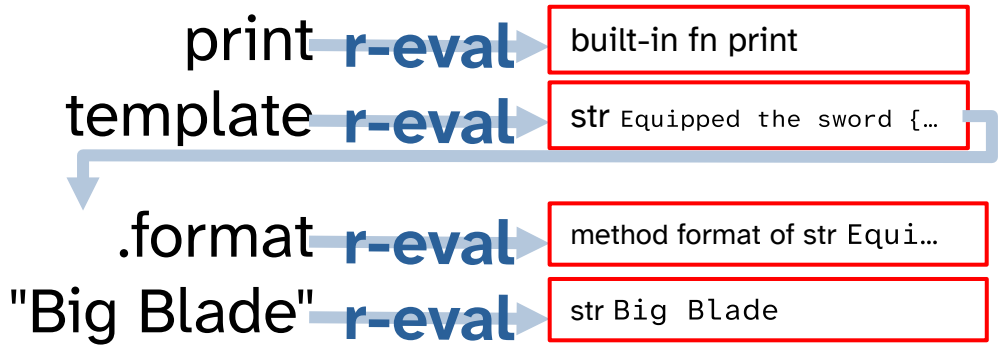


str Equipped…
has a
method format

Back to this again

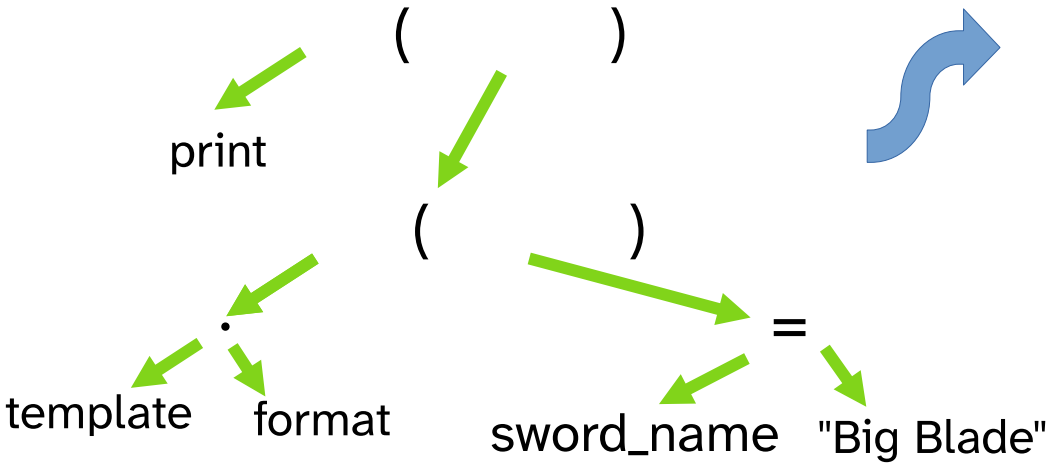
```
1    sword_name = "Big Blade"
2    template = 'Equipped the sword {sword_name}'
3    print(template.format(sword_name="Big Blade"))
```

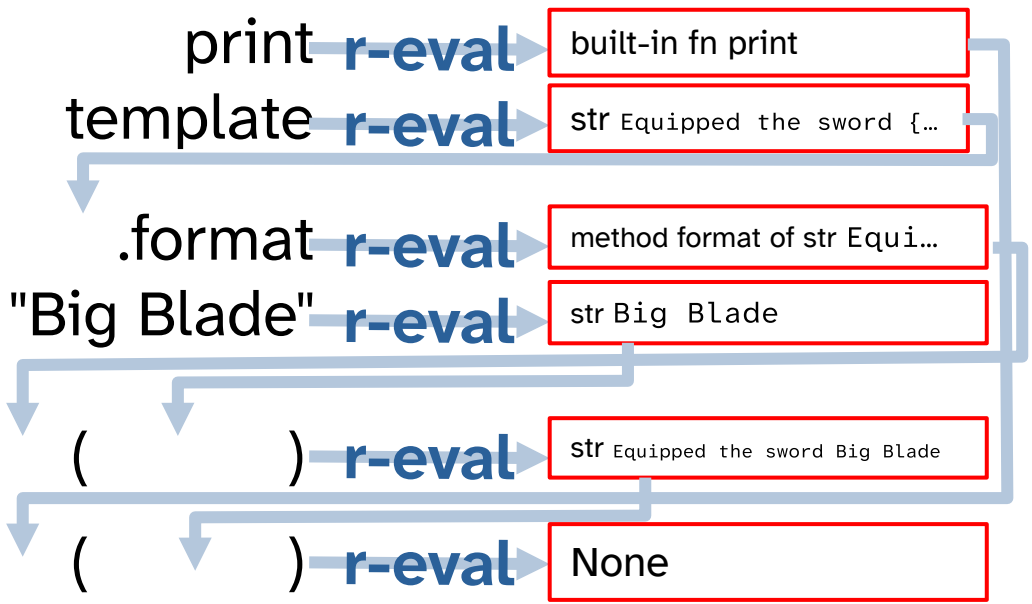




Back to this again

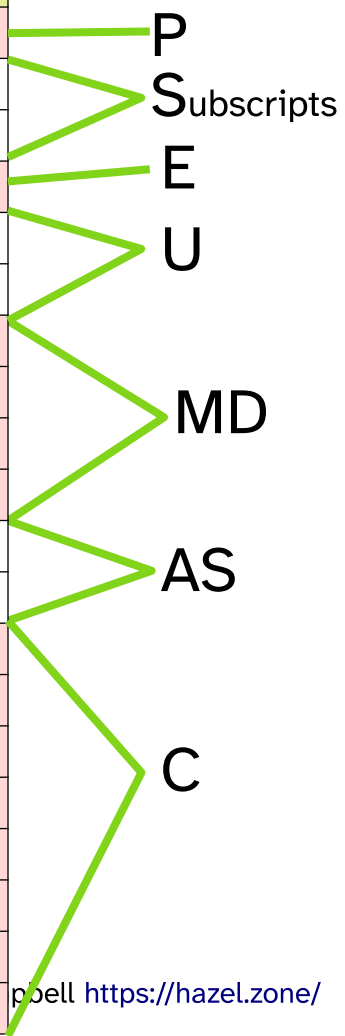
```
1   sword_name = "Big Blade"
2   template = 'Equipped the sword {sword_name}'
3   print(template.format(sword_name="Big Blade"))
```





Precedence

in type	characters	in type	result	operation
	( )			delimited expression
callable	( )			call
object	.	name		attribute reference
number	**	number	number	exponentiation
	+	number	number	positive
	-	number	number	negative / negation
number	*	number	number	multiplication
number	/	number	float	floating point division
number	//	number	number	floor division
number	%	number	number	remainder
number	+	number	number	addition
number	-	number	number	subtraction
number	<	number	bool	less than
number	<=	number	bool	less than equal
number	>	number	bool	greater than
number	>=	number	bool	greater than equal
number	==	number	bool	equal
number	!=	number	bool	unequal
object	is	object	bool	same object
object	is not	object	bool	different object



calls and . have the same operator precedence

(all operations that get something from something else)

Precedence

in type	characters	in type	result	operation
	( )		object	delimited expression
callable	( )		object	call
object	.	name	object	attribute reference
number	**	number	number	exponentiation
	+	number	number	positive
	-	number	number	negative / negation
number	*	number	number	multiplication
number	/	number	float	floating point division
number	//	number	number	floor division
number	%	number	number	remainder
number	+	number	number	addition
number	-	number	number	subtraction
number	<	number	bool	less than
number	<=	number	bool	less than equal
number	>	number	bool	greater than
number	>=	number	bool	greater than equal
number	==	number	bool	equal
number	!=	number	bool	unequal
object	is	object	bool	same object
object	is not	object	bool	different object
l-value	=	object		assignment

P Subscripts

E

U

MD

AS

C

A

calls and . have the same operator precedence

(all operations that get something from something else)