Deck for Lesson 008
Python – Tuples

Dr. Hazel “twitch.tv/hazeldotzone” Campbell

Tuples

- A Tuple in Python is delimited by (and) with a ,

lesson008_02.py

```
1   print("hello") # <---- parens make a call
2   print("hello", 2) # <- parens and , make a call
3   a = (5 + 2) * 3 # <--- parens make a delimited expression
4   print(a, type(a)) # <- parens and , make a call
5   t = (5, 2) # <-------- parens and , make a tuple
6   print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

Parentheses Rules

lesson008_02.py

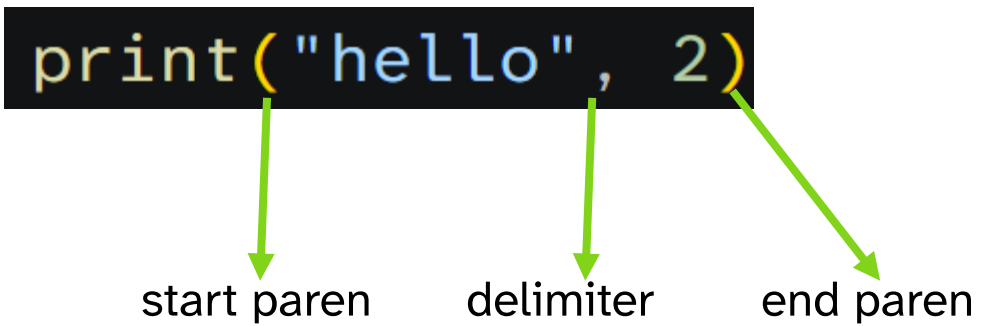
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

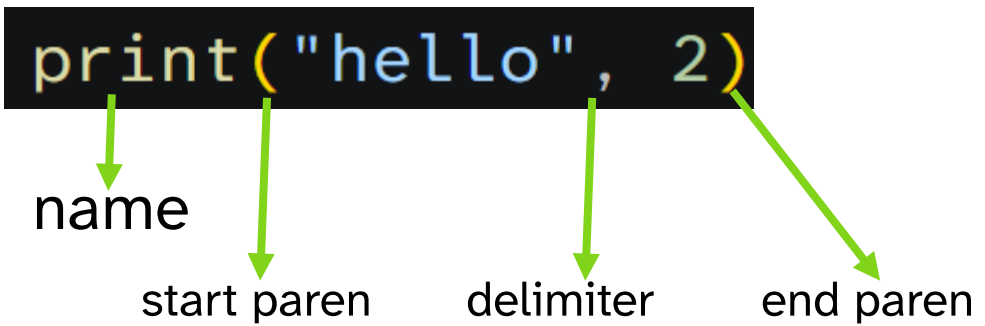
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

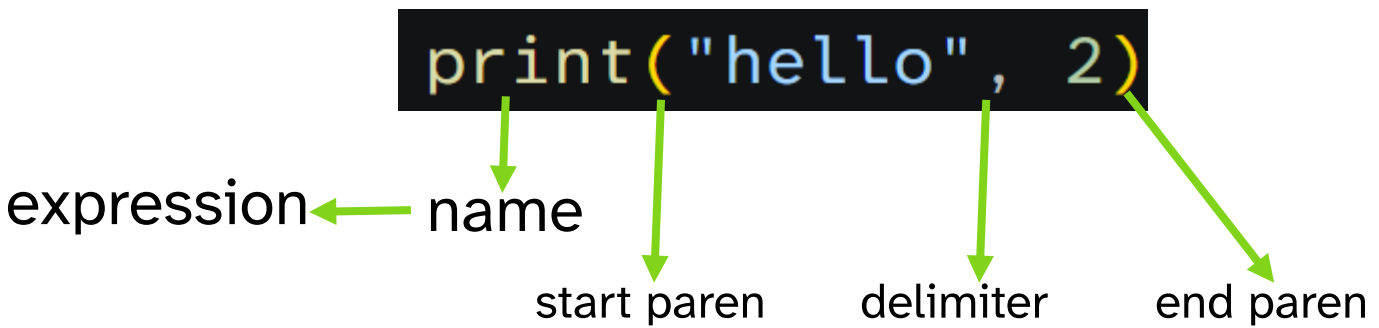
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

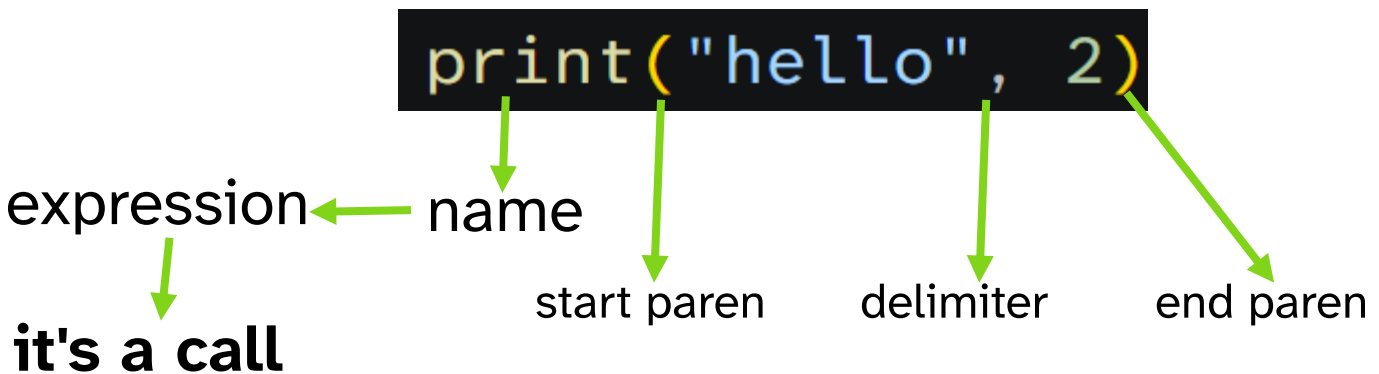
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

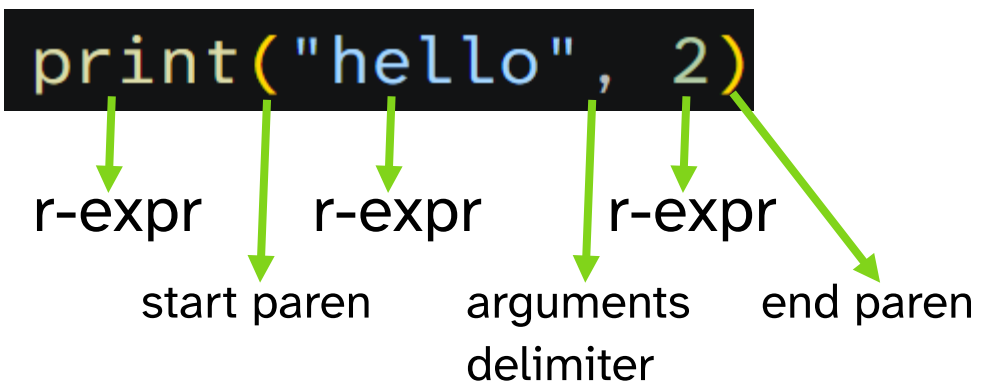
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

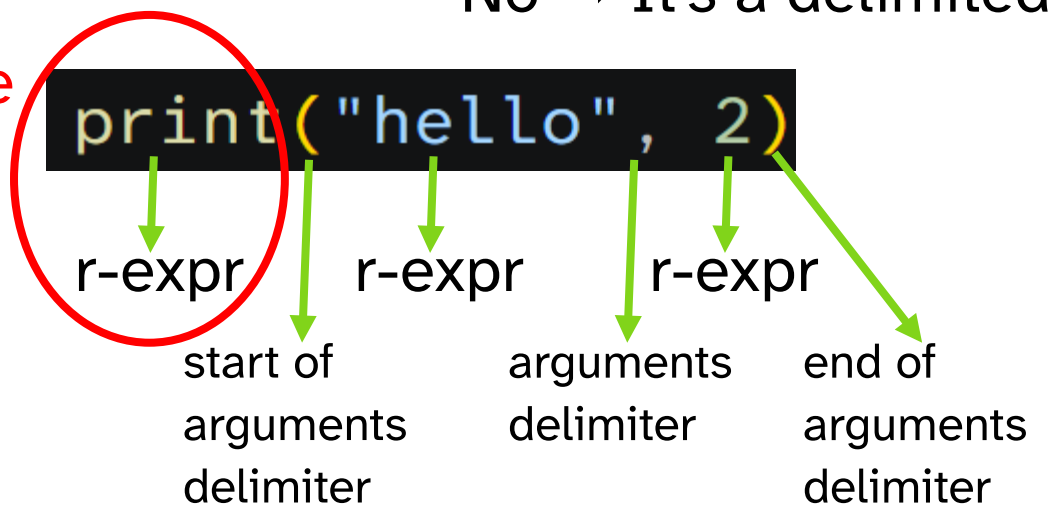
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

This is the thing that makes it a call!



Parentheses Rules

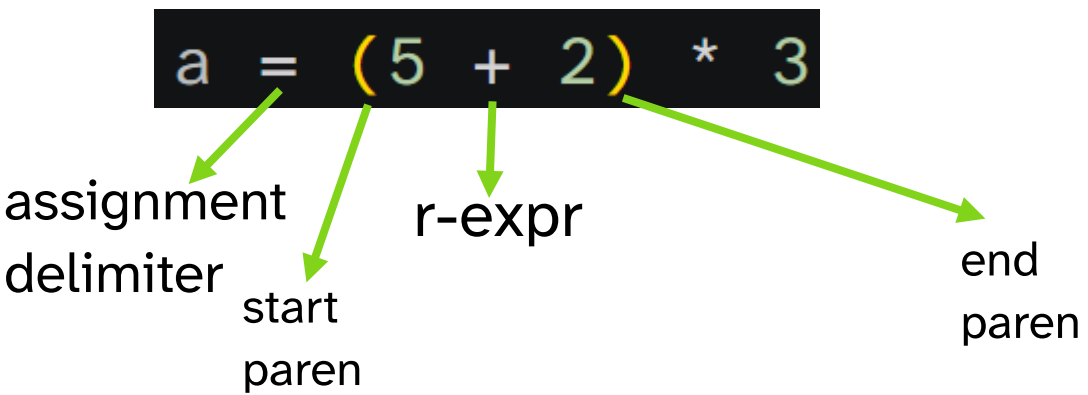
```
lesson008_02.py
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

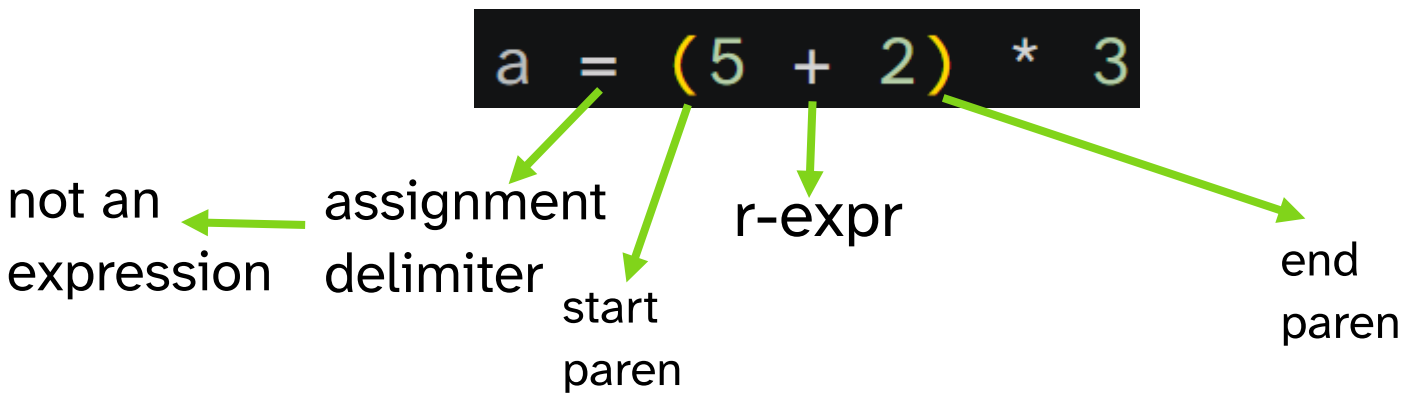
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

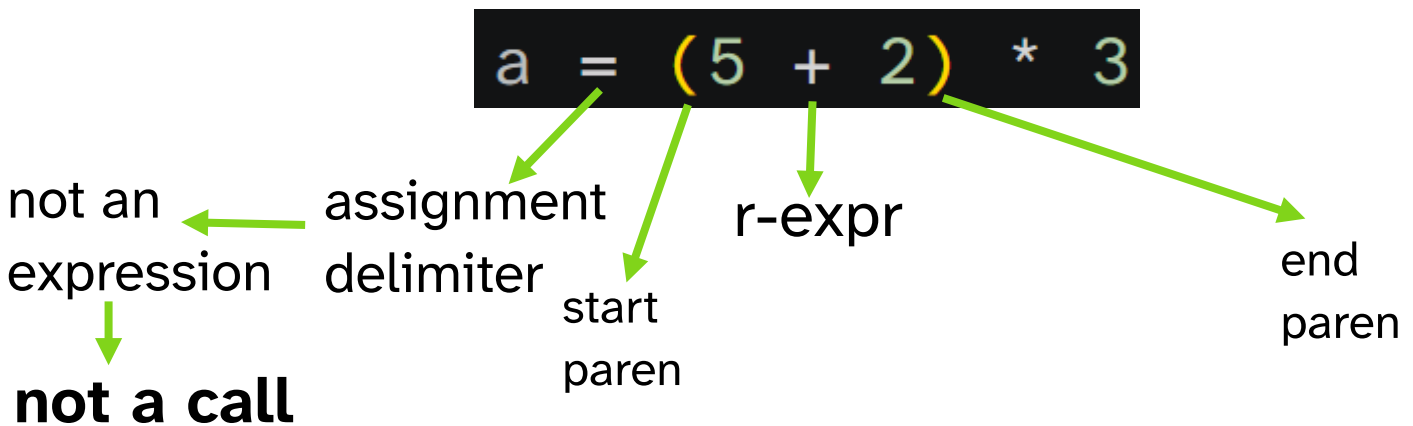
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

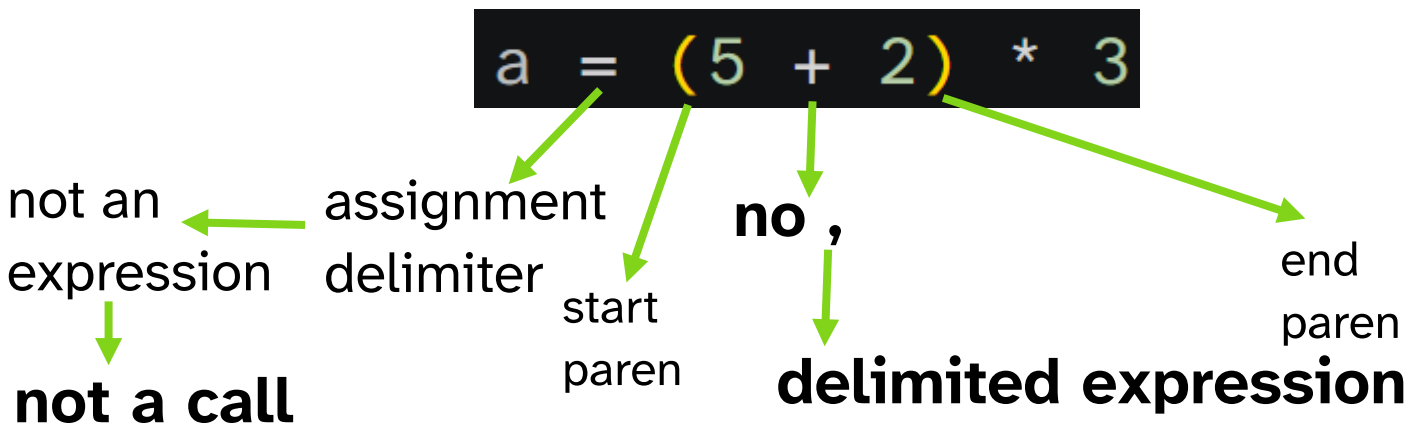
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

```
lesson008_02.py
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

not an expression → **not a call**

assignment delimiter

start paren

no ,

end paren

delimited expression

evaluates to whatever the expr between (and) evaluated to

Parentheses Rules

lesson008_02.py

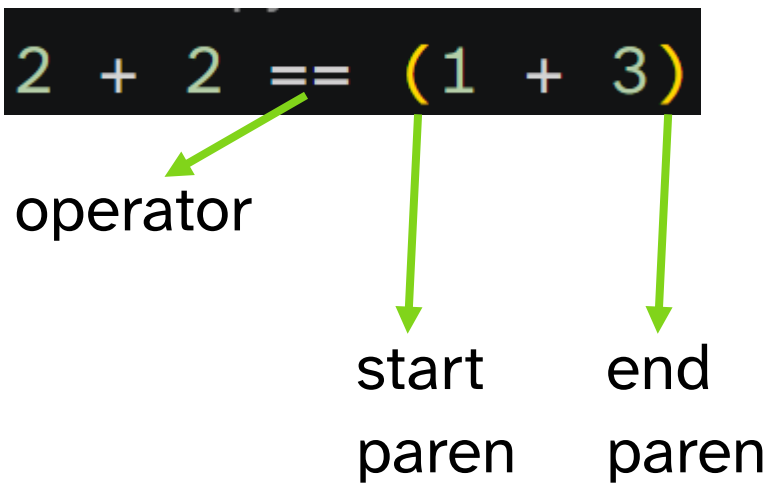
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

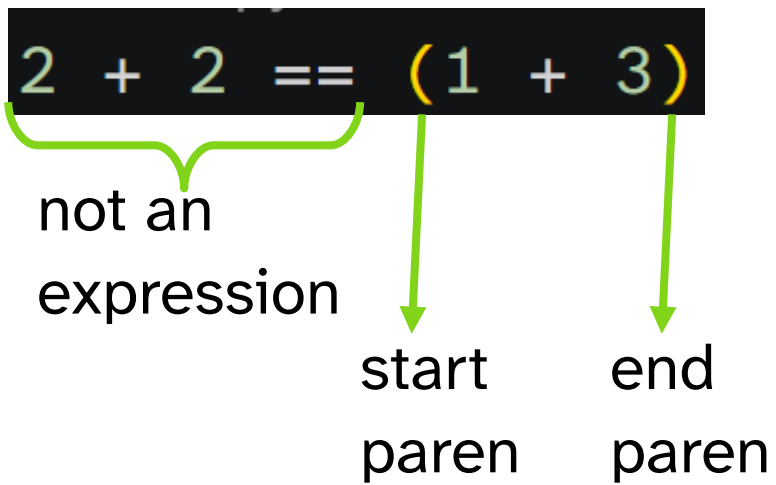
```
lesson008_02.py
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

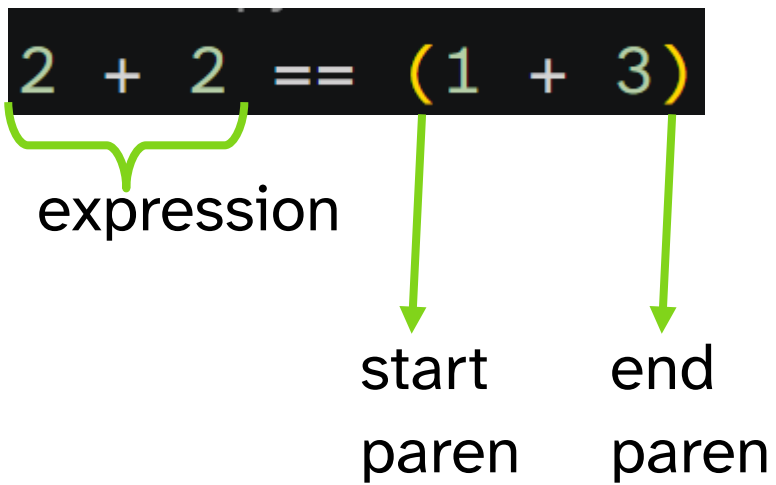
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
2 + 2 ==
```

not an expression

```
  File "C:\Users\hazeldotzone\Pyt
    (2 + 2 == )(1 + 3)
                ^
SyntaxError: invalid syntax
```

Parentheses Rules

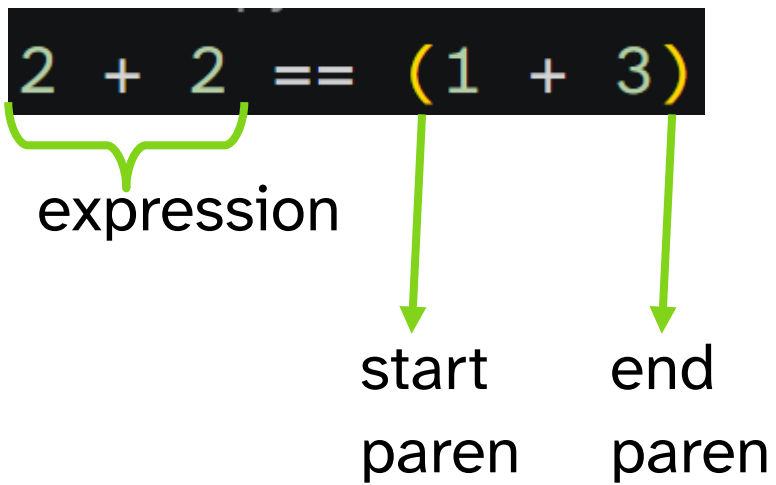
```
lesson008_02.py
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

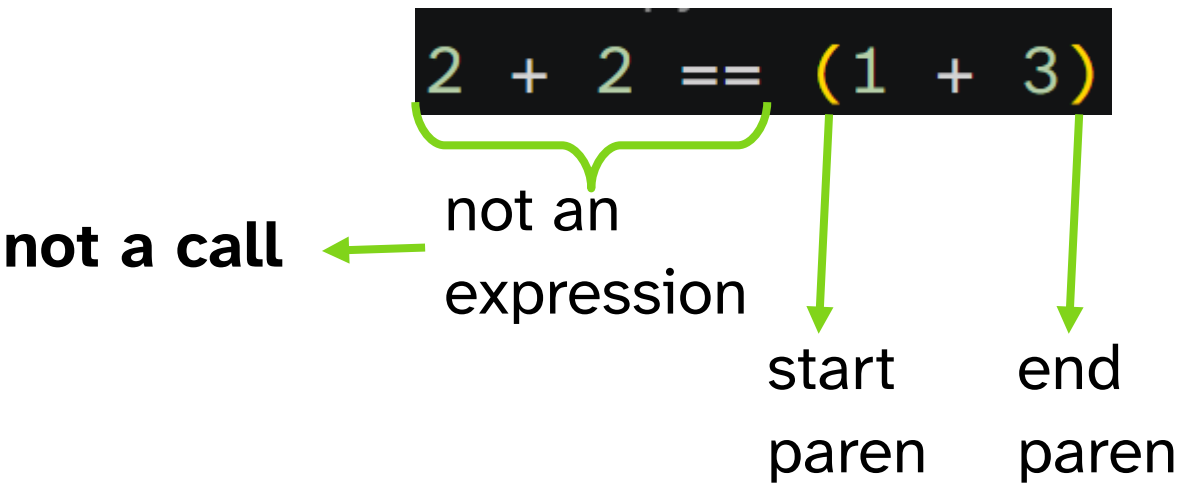
```
lesson008_02.py
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

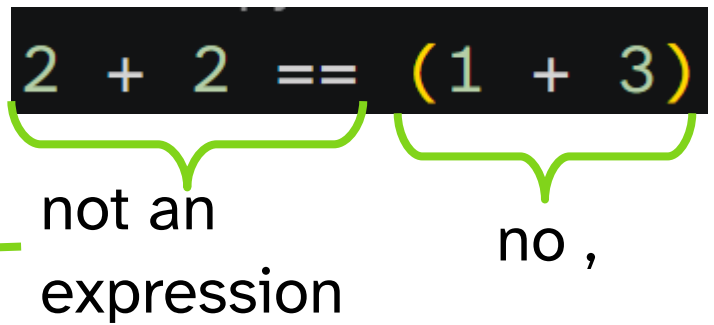
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

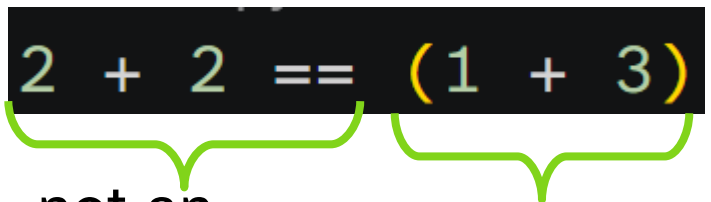
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

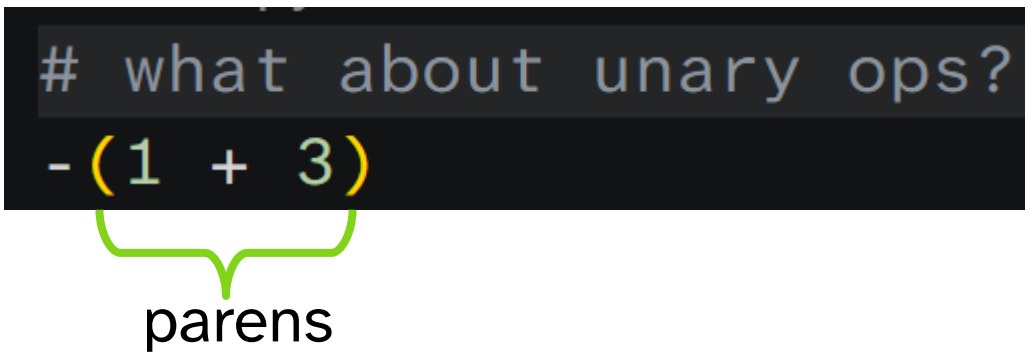
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

```
lesson008_02.py
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
# what ab
-(1 + 3)
```

parens

```
(-)(1 + 3)
 ^
SyntaxError: invalid syntax
```

Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
# what ab
-(1 + 3)
```

not an expr

parens

lesson008_25.py

```
1  # what about unary ops?
2  -
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_25.py
  File "C:\Users\hazeldotzone\Python Code\lesson008_25.py", line 2
    -
     ^
SyntaxError: invalid syntax
PS C:\Users\hazeldotzone\Python Code>
```

Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
# what about unary ops?
-(1 + 3)
```

not an expr

not a call

parens

Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
# what about unary ops?
-(1 + 3)
```

not an expr → **not a call**

parens

no , → **it's a delimited expression**

Parentheses Rules

```
lesson008_02.py
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
# what about this?
5(1+2)
```

parens

Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
# what about this?
5(1+2)
```

expression ← int literal

parens

Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
# what about this?
5(1+2)
```

it's a call

expression ← int literal

parens

Parentheses Rules

```
Traceback (most recent call last):
  File "C:\Users\hazeldotzone\Python Code\lesson008_25.py",
    5(1+2)
TypeError: 'int' object is not callable
```

```
# what about this?
5(1+2)
```

it's a call

expression ← int literal

parens

Python tries to call it in fact, but you can't call an int object, only the int type (class)

Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
t = (5, 2)
```

start paren

end paren

Parentheses Rules

lesson008_02.py

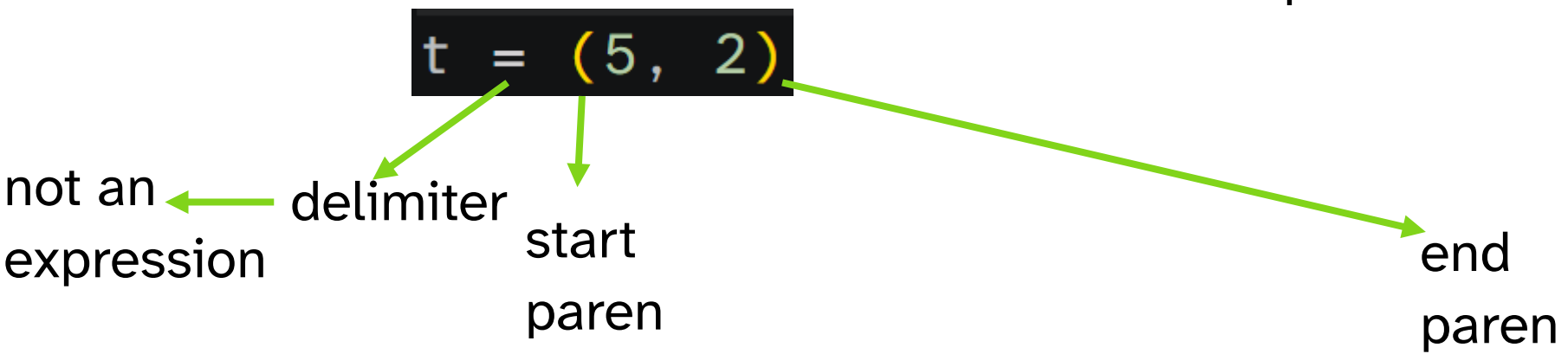
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
t = (5, 2)
```

- delimiter
  - not an expression → **not a call**
- start paren
- end paren

Parentheses Rules

lesson008_02.py

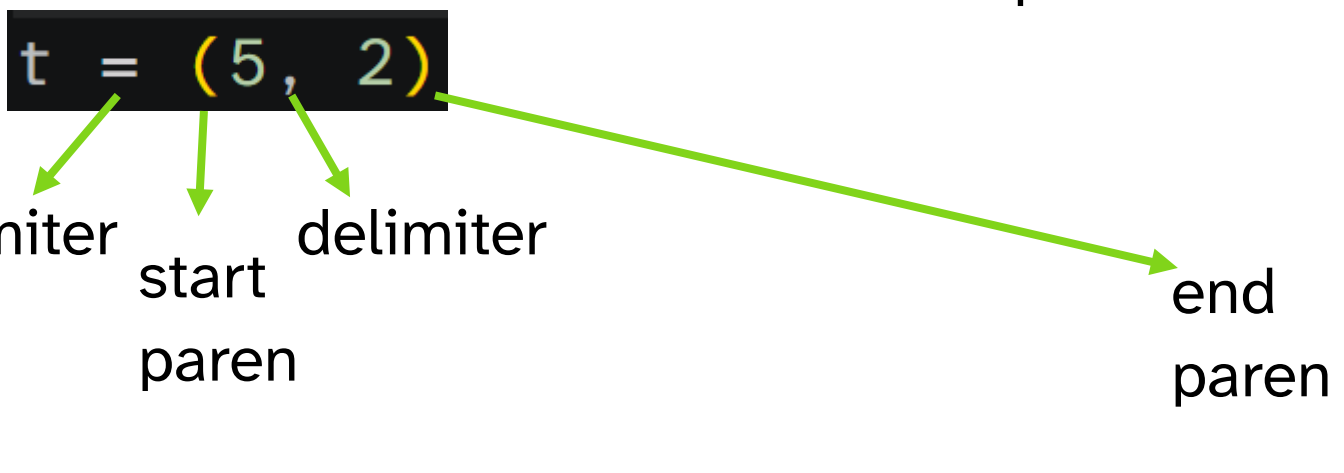
```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

lesson008_02.py

```
1  print("hello") # <---- parens make a call
2  print("hello", 2) # <- parens and , make a call
3  a = (5 + 2) * 3 # <--- parens make a delimited expression
4  print(a, type(a)) # <- parens and , make a call
5  t = (5, 2) # <-------- parens and , make a tuple
6  print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a ,
  - Yes → It's a tuple
  - No → It's a delimited expression

```
t = (5, 2)
```

- delimiter
  - not an expression → **not a call**
- start paren
- it's a comma , → **tuple!!**
- end paren

Displays

lesson008_37.py

```
1    a = 1
2    b = 2
3    t = (a, b)
4    u = (1, 2)
5    print(t == u)
```

tuple display
aka
tuple comprehension

Tuple "literals" aren't called "literals"
they are called
displays or *comprehensions*

Tuples Memory

lesson008_02.py

```
1   print("hello") # <---- parens make a call
2   print("hello", 2) # <- parens and , make a call
3   a = (5 + 2) * 3 # <---- parens make a delimited expression
4   print(a, type(a)) # <- parens and , make a call
5   t = (5, 2) # <-------- parens and , make a tuple
6   print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

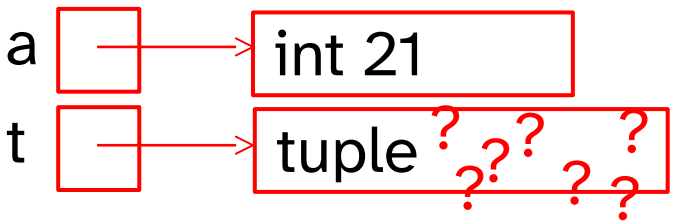
Tuples Memory

lesson008_02.py

```
1   print("hello") # <---- parens make a call
2   print("hello", 2) # <- parens and , make a call
3   a = (5 + 2) * 3 # <--- parens make a delimited expression
4   print(a, type(a)) # <- parens and , make a call
5   t = (5, 2) # <-------- parens and , make a tuple
6   print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```



Tuples Memory

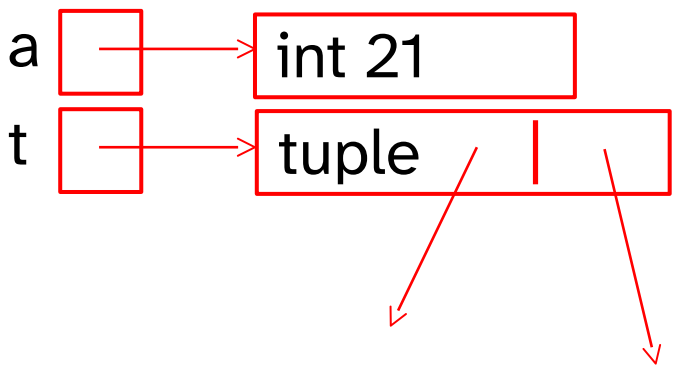
lesson008_02.py

```
1   print("hello") # <---- parens make a call
2   print("hello", 2) # <- parens and , make a call
3   a = (5 + 2) * 3 # <--- parens make a delimited expression
4   print(a, type(a)) # <- parens and , make a call
5   t = (5, 2) # <-------- parens and , make a tuple
6   print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

This tuple has 2 things it refers to



Tuples Memory

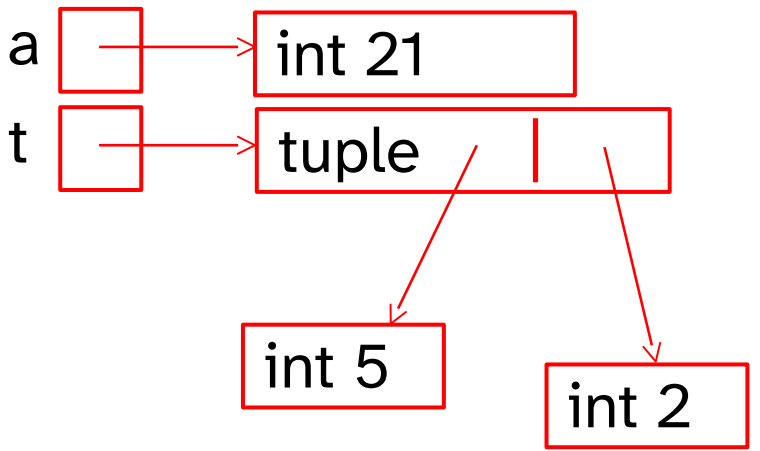
lesson008_02.py

```
1   print("hello") # <---- parens make a call
2   print("hello", 2) # <- parens and , make a call
3   a = (5 + 2) * 3 # <--- parens make a delimited expression
4   print(a, type(a)) # <- parens and , make a call
5   t = (5, 2) # <-------- parens and , make a tuple
6   print(t, type(t))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_02.py
hello
hello 2
21 <class 'int'>
(5, 2) <class 'tuple'>
PS C:\Users\hazeldotzone\Python Code>
```

This tuple has 2 things it refers to



Bigger Tuples

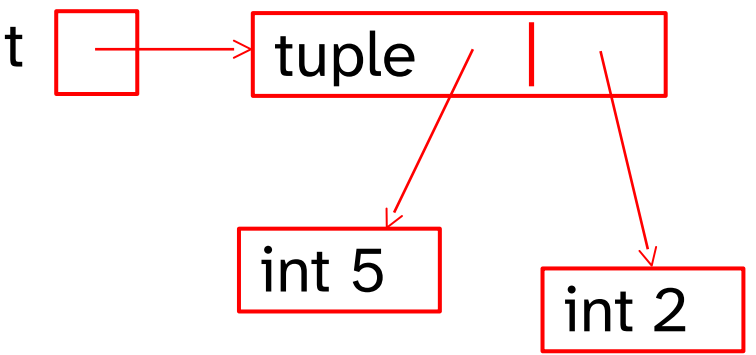
lesson008_41.py

```
1   t = (5, 2)
2   print(t)
3   print(type(t))
4   print(len(t))
5
```

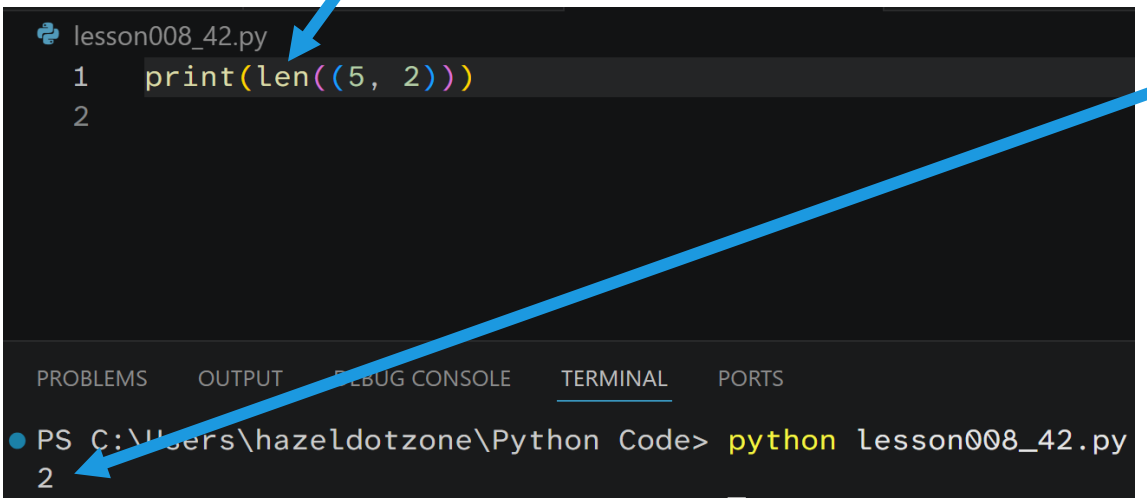
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_41.py
(5, 2)
<class 'tuple'>
2
PS C:\Users\hazeldotzone\Python Code>
```

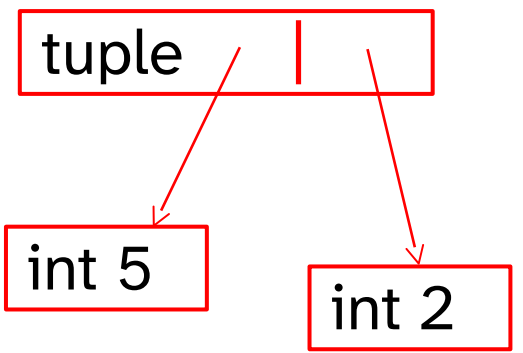
This tuple has 2 things it refers to



Bigger Tuples



This tuple has 2 things it refers to



of course, we don't have to assign the tuple but it's harder to read

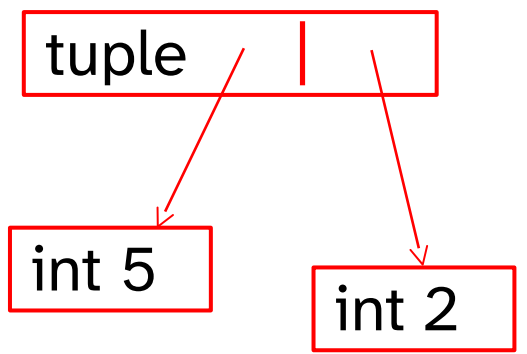
Bigger Tuples

```
print(len((5, 2)))
```

tuple expression

call

call



of course, we don't have to assign the tuple
but it's harder to read

Bigger Tuples

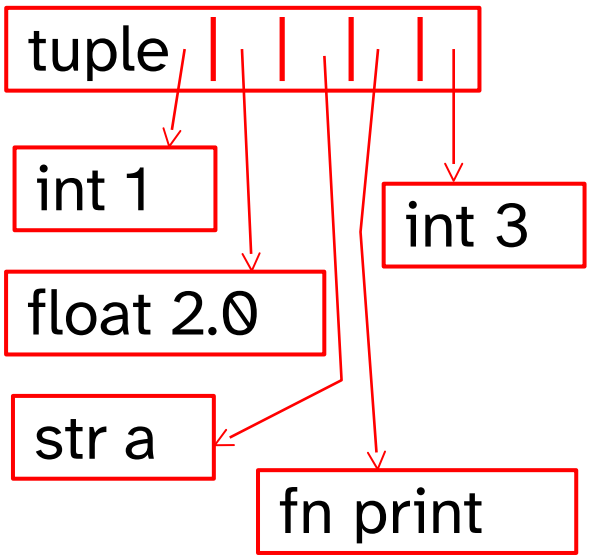
lesson008_42.py

```
1   t = (1, 2.0, "a", print, 1+2)
2   print(type(t), len(t))
3   print(str(t))
4
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_42.py
<class 'tuple'> 5
(1, 2.0, 'a', <built-in function print>, 3)
```

This tuple has 5 things it refers to



Bigger Tuples

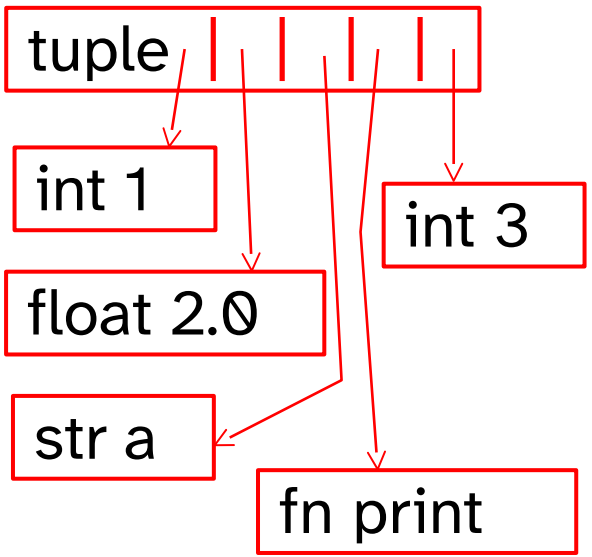
lesson008_42.py

```
1   t = (1, 2.0, "a", print, 1+2)
2   print(type(t), len(t))
3   print(str(t))
4
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_42.py
<class 'tuple'> 5
(1, 2.0, 'a', <built-in function print>, 3)
```

This tuple has 5 things it refers to



Bigger Tuples

lesson008_42.py

```
1   t = (1, 2.0, "a", print, 1+2)
2   print(type(t), len(t))
3   print(str(t))
4
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_42.py
<class 'tuple'> 5
(1, 2.0, 'a', <built-in function print>, 3)
```

Tuples & `str` or `repr`

lesson008_47.py

```
1   t = (1, 2.0, "a", print, 1+2)
2   print(type(t), len(t))
3   print(str(t))
4   print(repr(t))
5   print(str(t) == repr(t))
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_47.py
<class 'tuple'> 5
(1, 2.0, 'a', <built-in function print>, 3)
(1, 2.0, 'a', <built-in function print>, 3)
True
```

For tuples, like ints
`str`
does the same thing as
`repr`

Tuples & `str` or `repr`

lesson008_47.py

```
1   t = (1, 2.0, "a", print, 1+2)
2   print(type(t), len(t))
3   print(str(t))
4   print(repr(t))
5   print(str(t) == repr(t))
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_47.py
<class 'tuple'> 5
(1, 2.0, 'a', <built-in function print>, 3)
(1, 2.0, 'a', <built-in function print>, 3)
True
```

same as `repr("a")`

For tuples, like ints

`str`

does the same thing as

`repr`

tuple()

For int, str, and float we can call int() str() or float() to convert them...
What about tuple()?

```
>>> tuple(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>> tuple(2.0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object is not iterable
```

can't convert int to tuple

can't convert float to tuple

tuple()

For int, str, and float we can call int() str() or float() to convert them... What about tuple()?

```
>>> tuple("hello")
('h', 'e', 'l', 'l', 'o')
```

We can convert a string to a tuple... it will be the tuple with each letter separated out.

Unpacking

lesson008_51.py

```
1   (a, b, c) = (1, 2, 3)
2   print(f"a->{a}")
3   print(f"b->{b}")
4   print(f"c->{c}")
5
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_51.py
a->1
b->2
c->3
```

Unpacking

lesson008_51.py

```
1   (a, b, c) = (1, 2, 3)
```

let a refer to the first element
let b refer to the second element
let c refer to the third element

element of what? of whatever the right side evaluates to

Unpacking

lesson008_51.py

```
1   (a, b, c) = (1, 2, 3)
```

target list
l-expression

tuple
r-expression

As always with assignment,
it has the lowest precedence

Unpacking

lesson008_54.py

```
1   a, b, c = 1, 2, 3 # actually we don't even need the parens
2   print(f"a->{a}")
3   print(f"b->{b}")
4   print(f"c->{c}")
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson008_54.py
a->1
b->2
c->3
PS C:\Users\hazeldotzone\Python Code>
```

Unpacking

lesson008_55.py

```
1   a = 1, 2, 3 # actually we don't even need the parens
2   print(f"a->{a}")
3
4
```

We can omit the parens in some situations (when it's unambiguous what's going on)

PROBLEMS OUTPUT DEBUG CONS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson008_55.py
a->(1, 2, 3)
PS C:\Users\hazeldotzone\Python Code>
```

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)

Unpacking

As always with assignment,
it has the lowest precedence

lesson008_51.py

```
1   (a, b, c) = (1, 2, 3)
```

target list
l-expression

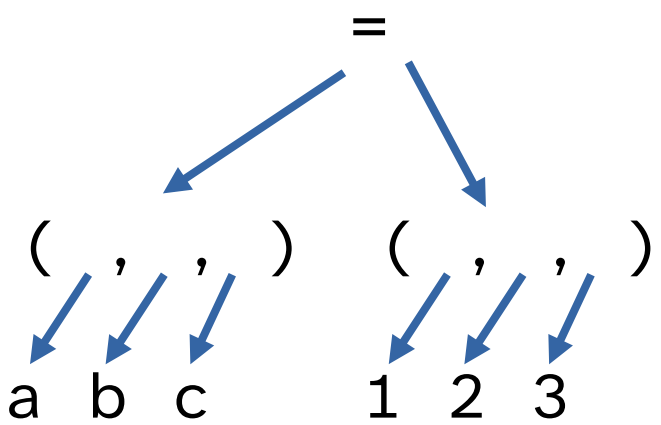
tuple
r-expression

Unpacking

As always with assignment, it has the lowest precedence

lesson008_51.py

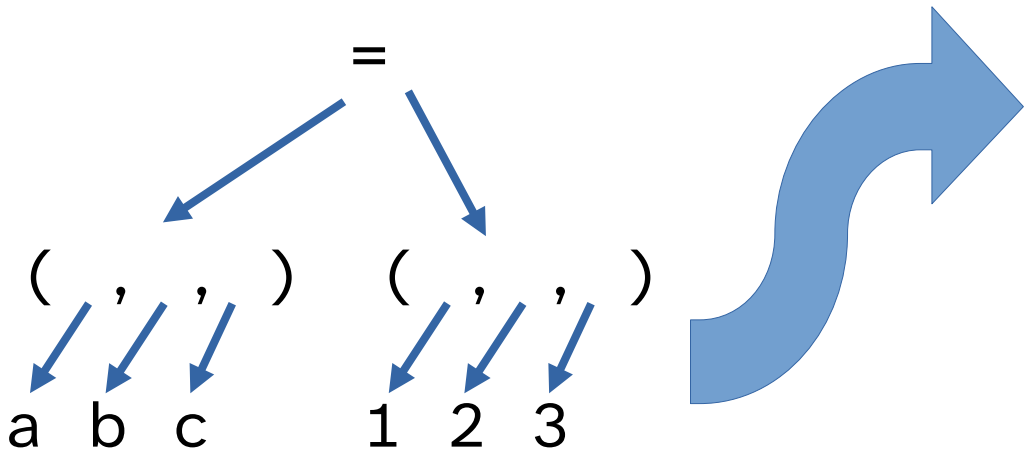
```
1   (a, b, c) = (1, 2, 3)
```



Unpacking

As always with assignment,
it has the lowest precedence

```
lesson008_51.py
1   (a, b, c) = (1, 2, 3)
```



find or create a

find or create b

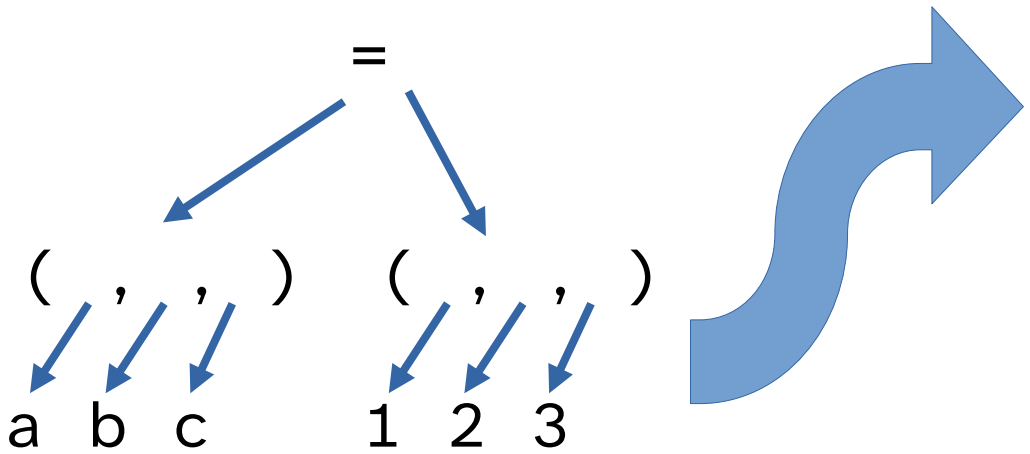
find or create c

Unpacking

As always with assignment,
it has the lowest precedence

lesson008_51.py

```
1   (a, b, c) = (1, 2, 3)
```



find or create a

find or create b

find or create c

literal 1 → int 1

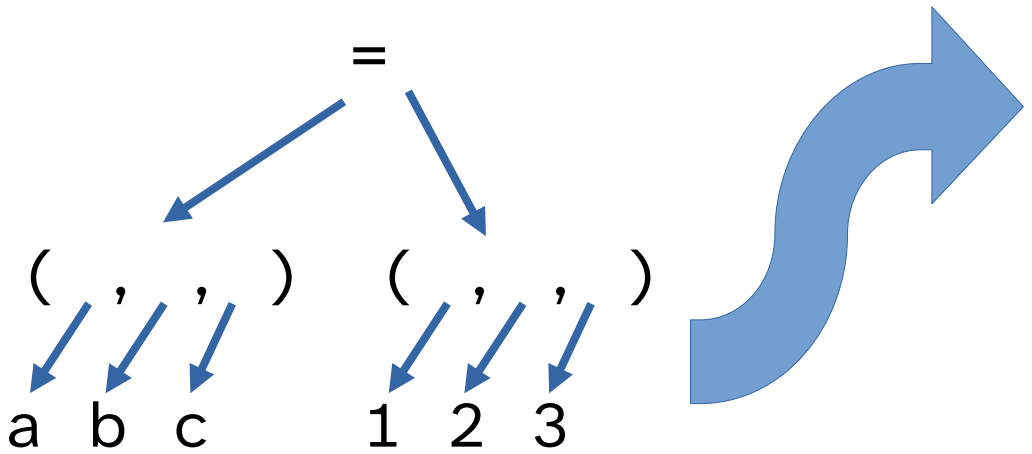
literal 2 → int 2

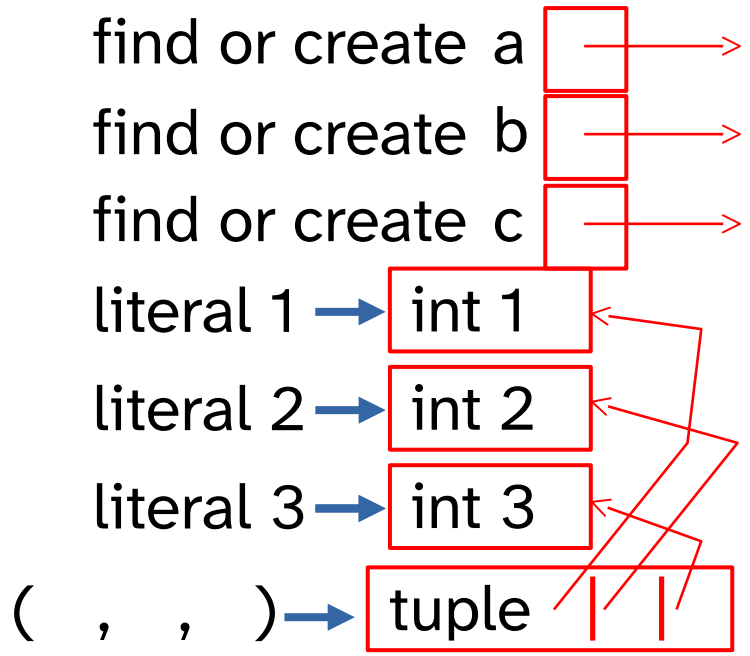
literal 3 → int 3

Unpacking

As always with assignment,
it has the lowest precedence

```
lesson008_51.py
1   (a, b, c) = (1, 2, 3)
```



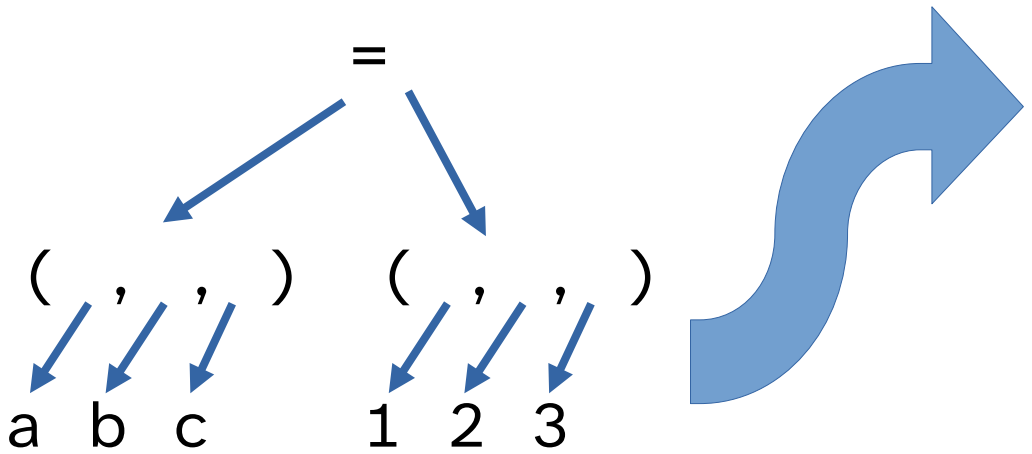


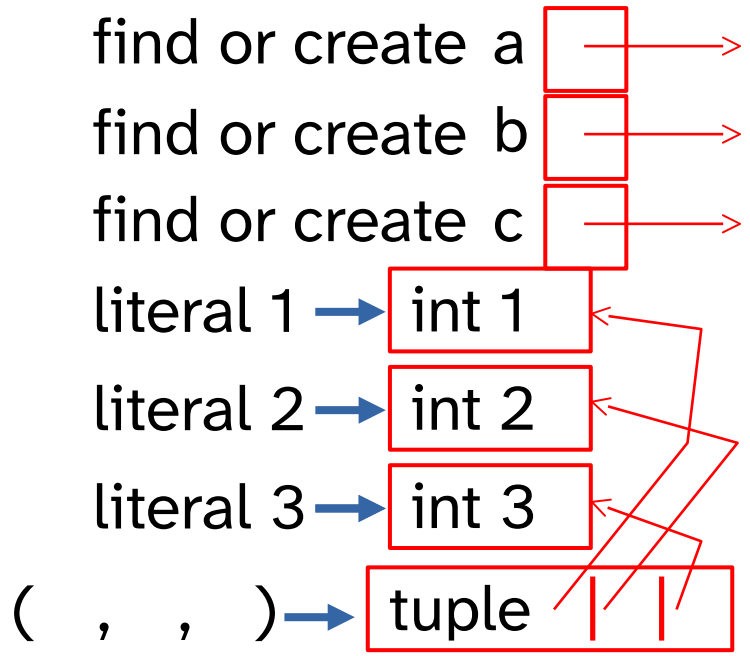
Unpacking

As always with assignment,
it has the lowest precedence

lesson008_51.py

```
1   (a, b, c) = (1, 2, 3)
```



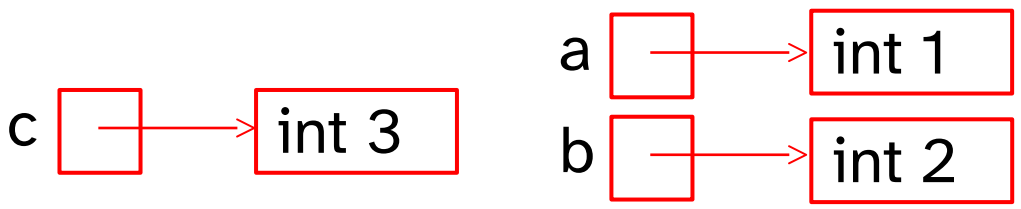


Unpacking

As always with assignment,
it has the lowest precedence

lesson008_51.py

```
1   (a, b, c) = (1, 2, 3)
```



Unpacking

As always with assignment,
it has the lowest precedence

lesson008_51.py

```
1   (a, b, c) = (1, 2, 3)
```

We should only use unpacking when:

1. we already have a tuple
2. it makes things easier to read

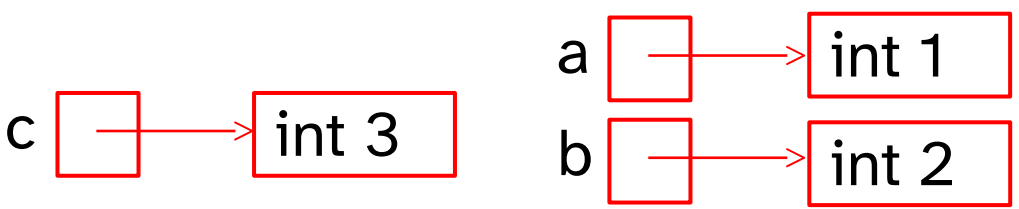
lesson008_63.py

```
1   t = (1, 2, 3)
2   # ... a bunch of code omitted ...
3   # ... some time later ...
4   (a, b, c) = t
5   print(f"a->{a} b->{b} c->{c}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_63.py
a->1 b->2 c->3
PS C:\Users\hazeldotzone\Python Code>
```

1. we already have a tuple



Unpacking

```
(x, y, z) = (0.0, 0.0, 0.0) # start at the origin
```

2. it makes things easier to read

Smaller Tuples

lesson008_64.py

```
1   t = (77,) # tuple with one element. It still has to have a ,
2   print(f"type: {type(t)}, len: {len(t)}, repr: {t!r}")
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_64.py
type: <class 'tuple'>, len: 1, repr: (77,)
PS C:\Users\hazeldotzone\Python Code>
```

Smaller Tuples

lesson008_65.py

```
1   t = (77) # not a tuple, just a delimited expression
2   print(f"type: {type(t)}, len: {len(t)}, repr: {t!r}")
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_65.py
Traceback (most recent call last):
  File "C:\Users\hazeldotzone\Python Code\lesson008_65.py", line 2, in <module>
    print(f"type: {type(t)}, len: {len(t)}, repr: {t!r}")
                           ^^^^^^
TypeError: object of type 'int' has no len()
PS C:\Users\hazeldotzone\Python Code>
```

Smaller Tuples

lesson008_66.py

```
1   t = (,77) # invalid syntax: the , has to go at the end
2   print(f"type: {type(t)}, len: {len(t)}, repr: {t!r}")
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_66.py
  File "C:\Users\hazeldotzone\Python Code\lesson008_66.py", line 1
    t = (,77) # tuple with one element. It still has to have a ,
         ^
SyntaxError: invalid syntax
PS C:\Users\hazeldotzone\Python Code>
```

Smaller Tuples

lesson008_68.py

```
1   t = () # create a tuple of length 0 ... no ,
2   print(f"type: {type(t)}, len: {len(t)}, repr: {t!r}")
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_68.py
type: <class 'tuple'>, len: 0, repr: ()
PS C:\Users\hazeldotzone\Python Code>
```

Parentheses Rules

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression

Parentheses Rules

Commas at the end are allowed, commas at the beginning arent

Does it have an expression before the (

- Yes → It's a call
- No → Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression

```
t = () # tuple
u = (1,) # tuple
v = (1,2,) # tuple
```

Parentheses Rules

Commas at the end are allowed, commas at the beginning aren't

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression

lesson008_71.py

```
1   a = (1, 2)
2   b = (1, 2, )
3   print(a == b)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_71.py
True
```

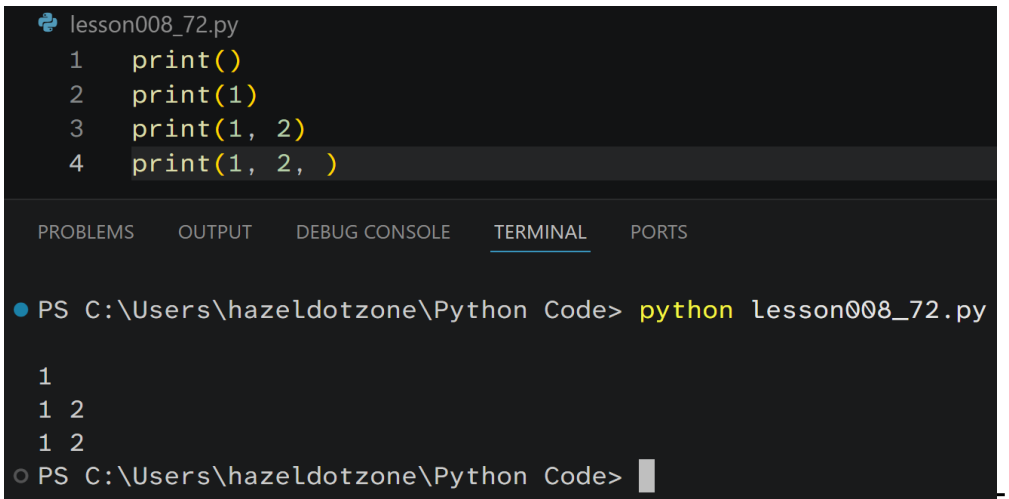
Parentheses Rules

Commas at the end are allowed, commas at the beginning aren't

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression

ok



Parentheses Rules

Commas at the end are allowed, commas at the beginning aren't

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression

ok

lesson008_73.py

```
1  print( # this is a call there's an expr before the ()
2      1,
3      2.0,
4      "three"
5  )
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_73.py
1 2.0 three
PS C:\Users\hazeldotzone\Python Code>
```

Parentheses Rules

Commas at the end are allowed, commas at the beginning arent

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression

ok

lesson008_73.py

```
print( # this is a call there's an expr before the ()
    1,
    2.0,
    "three"
)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_73.py
1 2.0 three
PS C:\Users\hazeldotzone\Python Code>
```

Parentheses Rules

Does it have an expression before the (

- Yes → It's a call
- No→Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression

lesson008_75.py

```
1   # remember if we want to call with a tuple,
2   # we have to put it in its own parens
3   print((1, 2.0, "three"))
4
```

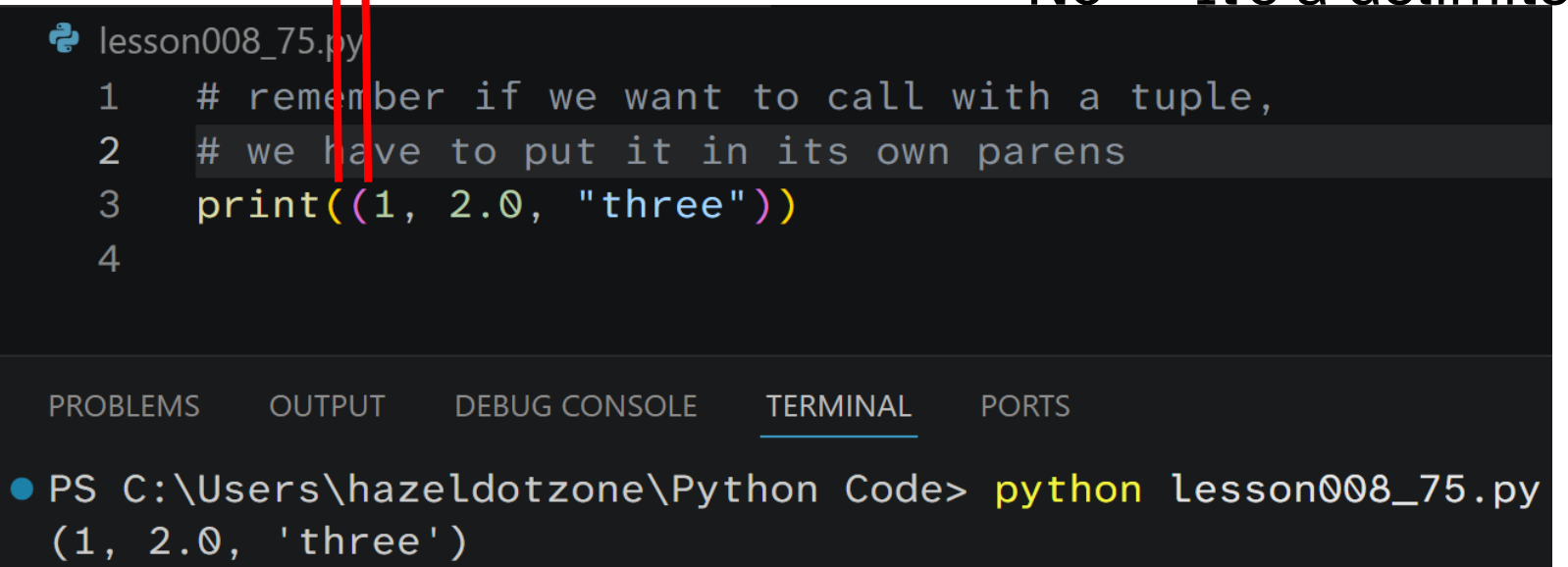
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson008_75.py
(1, 2.0, 'three')
```

Parentheses Rules

Does it have an expression before the (

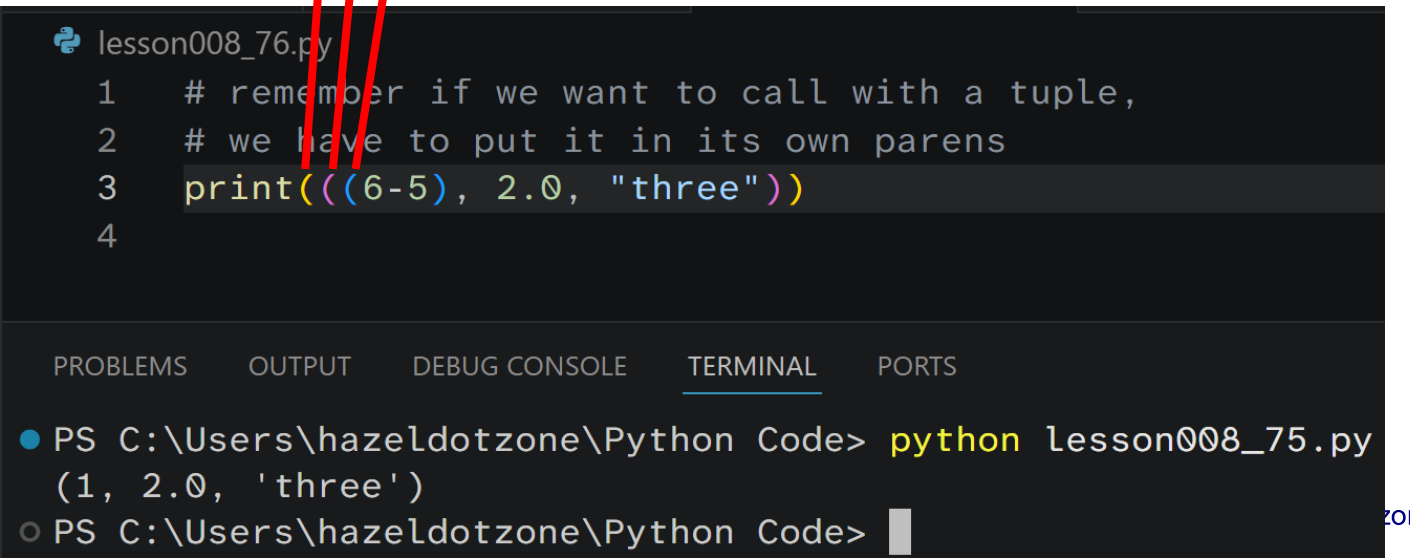
- Yes → It's a call
- No→Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression



Parentheses Rules

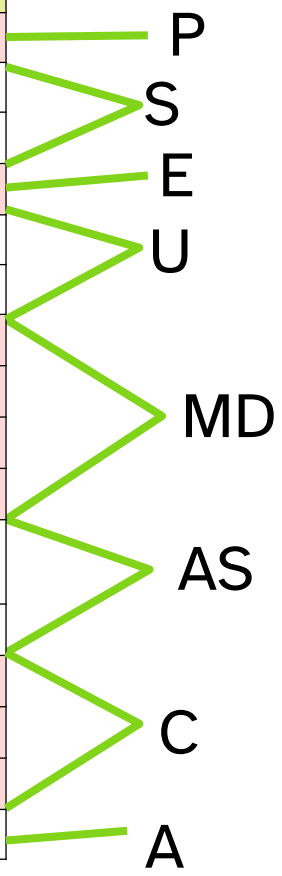
Does it have an expression before the (

- Yes → It's a call
- No→Does it have a , **or nothing ( )**
  - Yes → It's a tuple
  - No → It's a delimited expression



Precedence

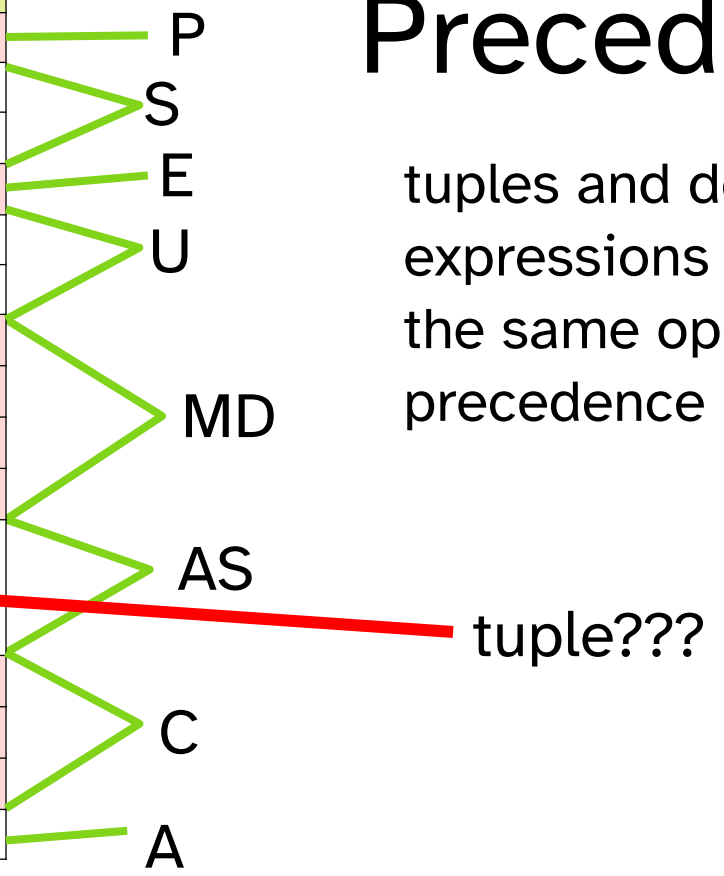
in type	characters	in type	result	operation
	( ) ,		object	delimited expression, tuple
callable	___( )		object	call
object	.	name	object	attribute reference
number	**	number	number	exponentiation
	+	number	number	positive
	-	number	number	negative / negation
number	*	number	number	multiplication
number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



tuples and delimited expressions have the same operator precedence

Precedence

in type	characters	in type	result	operation
	( ) ,		object	delimited expression, tuple
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 sequence	+	number sequence	number sequence	addition concatentation
number	-	number	number	subtraction
number	< <= > >=	number	bool	various comparisons
number	== !=	number	bool	equal, unequal
object	is is not	object	bool	same object
l-value	=	object		assignment



tuples and delimited expressions have the same operator precedence

Concatenation