Deck for Lesson 007
Python – Formatting

Dr. Hazel “twitch.tv/hazeldotzone” Campbell

The Build-A-String Workshop

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

So, We Converted It To a Formatted String (f-string)

```python
sword_name = "Big Blade"
sword_damage = 6
sword_weight = 2.1
sword_level = 3
print(f"{sword_name} is a sword that does {sword_damage} damage, weighs {sword_weight}kg and requires level {sword_level}.")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hazeldotzone\Python Code> python .\lesson007.py
Big Blade is a sword that does 6 damage, weighs 2.1kg and requires level 3.
PS C:\Users\hazeldotzone\Python Code>
```

Then We Figured Out Save and Use The Format like a Template

lesson007b.py

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007b.py
Big Blade is a sword that does 6 damage, weighs 2.1kg and requires level 3.
PS C:\Users\hazeldotzone\Python Code>
```

Then We Figured Out Save and Use The Format like a Template

lesson007b.py

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

Names corresponding to the {fields} in the string, **not** to variables!

Then We Figured Out Save and Use The Format like a Template

lesson007b.py

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

r-expressions that happen to consist of only a variable name, in this case variables that refer to

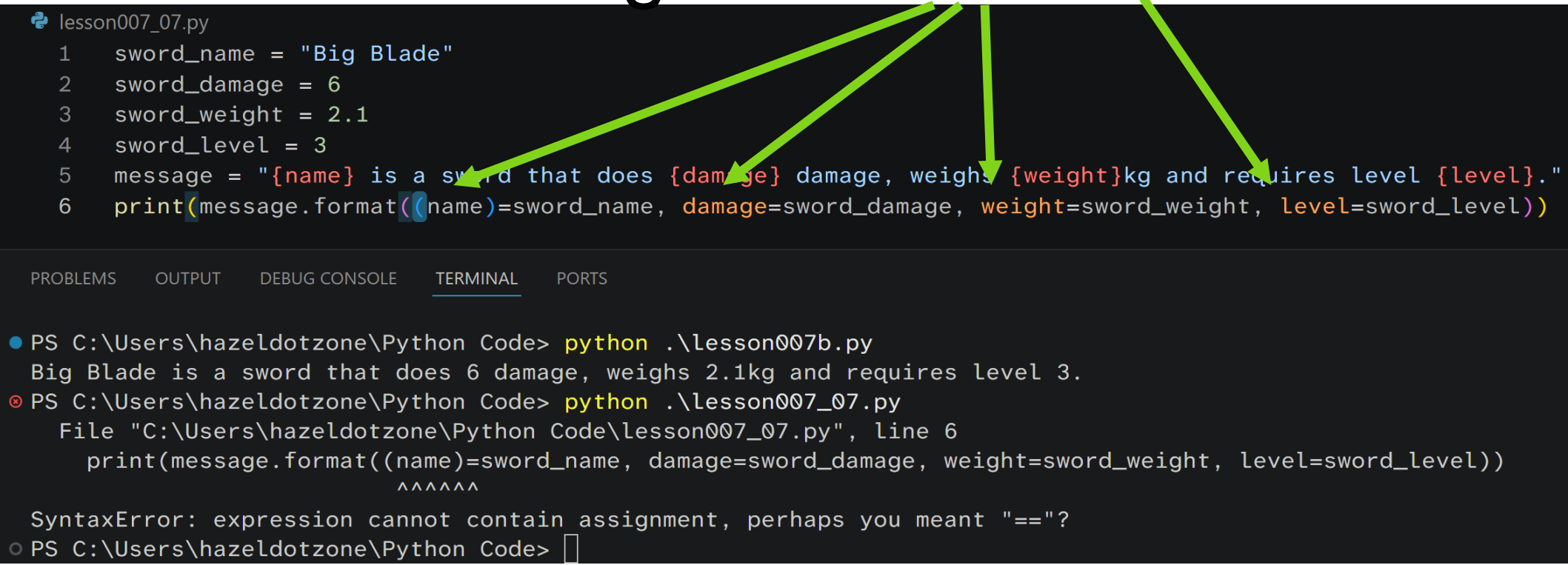
str `Big Blade`

int `6`

float 2.1

int `3`

We Can't Even Put Parens Around The Argument Names



But we can do any operations we want in argument expressions!

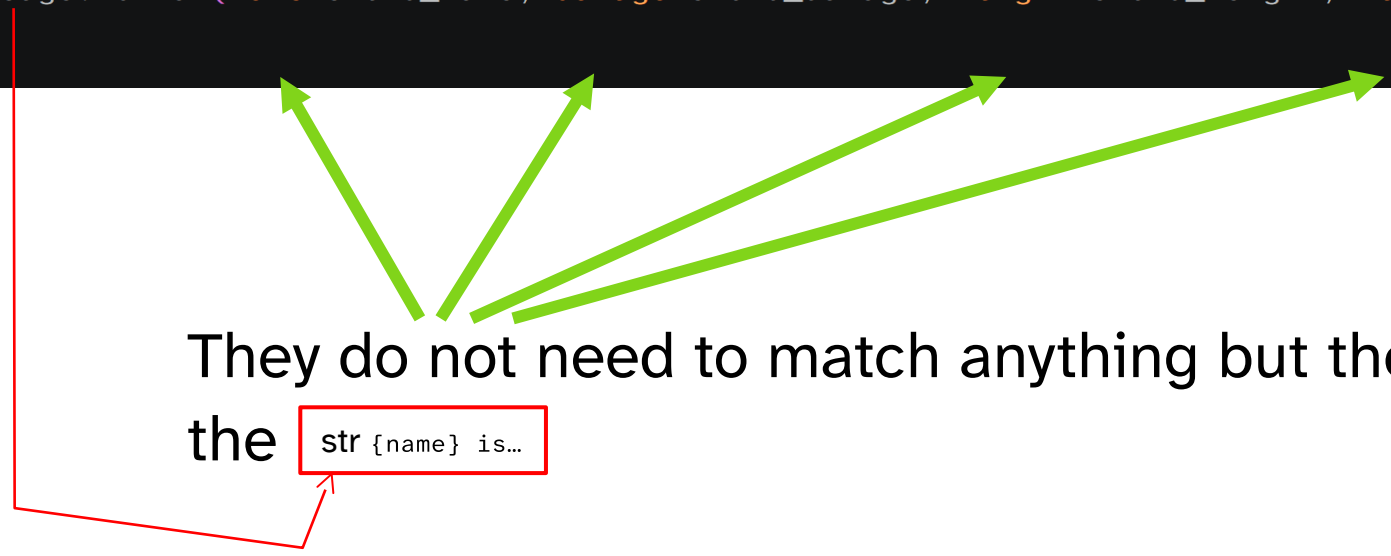
The errors are giving us important information!

- In this case, we can tell from the error we get when we try to put parens around the argument name, that it's not any kind of expression.

These argument names are a different kind of name.

lesson007b.py

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



They do not need to match anything but the fields in the `str {name} is…`

Or

lesson007_09.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6
3   sword_weight = 2.1
4   sword_level = 3
5   message = "{sword_name} is a sword that does {sword_damage} damage, weighs {sword_weight}kg and requires level {sword_level}."
6   print(message.format(sword_name=sword_name+" II", sword_damage=sword_damage//2, sword_weight=sword_weight+1, sword_level=(sword_level)))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_08.py
Big Blade II is a sword that does 3 damage, weighs 3.1kg and requires level 3.
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_09.py
Big Blade II is a sword that does 3 damage, weighs 3.1kg and requires level 3
PS C:\Users\hazeldotzone\Python Code>
```

They do not need to match anything but the fields in the `str {name} is…`

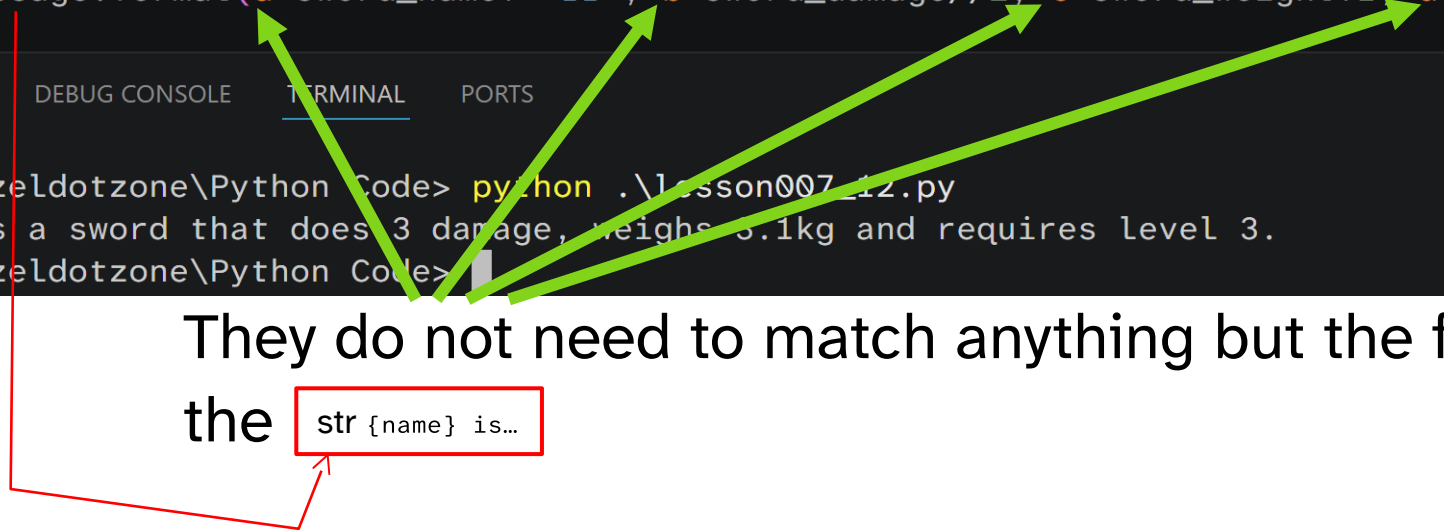
Or they can be whatever...

lesson007_12.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6
3   sword_weight = 2.1
4   sword_level = 3
5   message = "{a} is a sword that does {b} damage, weighs {c}kg and requires level {d}."
6   print(message.format(a=sword_name+" II", b=sword_damage//2, c=sword_weight+1, d=(sword_level)))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_12.py
Big Blade II is a sword that does 3 damage, weighs 3.1kg and requires level 3.
PS C:\Users\hazeldotzone\Python Code>
```



They do not need to match anything but the fields in the `str {name} is…`

Or they can be whatever...

lesson007_12.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6
3   sword_weight = 2.1
4   sword_level = 3
5   message = "{a} is a sword that does {b} damage, weighs {c}kg and requires level {d}."
6   print(message.format(a=sword_n                    ght+1, d=(sword_level)))
```

PROBLEMS OUTPUT

PS C:\U
Big Bl
PS C:\U

Of course, we would never want to use such bad names like a, b, c, d because they don't convey any information to other programmers or ourselves

...the fields in

Field Width

lesson007_14.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6000
3   sword_weight = 2.23606797749979
4   sword_level = 3
5   message = "{name:3} is a sword that does {damage:3} damage, weighs {weight:3}kg and requires level {level:3}."
6   print(message.format(name=sword_name, damage=sword_damage, weight=sword_weight, level=sword_level))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_14.py
Big Blade is a sword that does 6000 damage, weighs 2.23606797749979kg and requires level   3.
PS C:\Users\hazeldotzone\Python Code>
```

The :3 means "make this take up at least 3 characters"

Field Width

lesson007_15.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6000
3   sword_weight = 2.23606797749979
4   sword_level = 3
5   message = "{name:3} is a sword that does {damage:3} damage, weighs {weight:3}kg and requires level {level:03}."
6   print(message.format(name=sword_name, damage=sword_damage, weight=sword_weight, level=sword_level))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_14.py
Big Blade is a sword that does 6000 damage, weighs 2.23606797749979kg and requires level   3.
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_15.py
Big Blade is a sword that does 6000 damage, weighs 2.23606797749979kg and requires level 003.
PS C:\Users\hazeldotzone\Python Code>
```

The :03 means "make this take up at least 3 characters and use 0 for padding"

Field Width

lesson007_16.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6000
3   sword_weight = 2.23606797749979
4   sword_level = 3
5   message = "{name:3} is a sword that does {damage:3} damage, weighs {weight:3}kg and requires level {level:<3}."
6   print(message.format(name=sword_name, damage=sword_damage, weight=sword_weight, level=sword_level))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_14.py
Big Blade is a sword that does 6000 damage, weighs 2.23606797749979kg and requires level   3.
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_15.py
Big Blade is a sword that does 6000 damage, weighs 2.23606797749979kg and requires level 003.
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_16.py
Big Blade is a sword that does 6000 damage, weighs 2.23606797749979kg and requires level 3   .
PS C:\Users\hazeldotzone\Python Code>
```

The :<3 means "make this take up at least 3 characters and align to the left"

Field Options

lesson007_17.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6000
3   sword_weight = 2.23606797749979
4   sword_level = 3
5   message = "{name:3} is a sword that does {damage:,} damage, weighs {weight:3}kg and requires level {level:<3}."
6   print(message.format(name=sword_name, damage=sword_damage, weight=sword_weight, level=sword_level))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_17.py
Big Blade is a sword that does 6,000 damage, weighs 2.23606797749979kg and requires level 3  .
PS C:\Users\hazeldotzone\Python Code>
```

The :, means "add commas every 3 digits"

Field Options

lesson007_18.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6000
3   sword_weight = 2.23606797749979
4   sword_level = 3
5   message = "{name:3} is a sword that does {damage:_} damage, weighs {weight:3}kg and requires level {level:<3}."
6   print(message.format(name=sword_name, damage=sword_damage, weight=sword_weight, level=sword_level))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_17.py
Big Blade is a sword that does 6,000 damage, weighs 2.23606797749979kg and requires level 3  .
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_18.py
Big Blade is a sword that does 6_000 damage, weighs 2.23606797749979kg and requires level 3  .
PS C:\Users\hazeldotzone\Python Code>
```

The :_ means "add underscores every 3 digits"

Field Options

lesson007_19.py

```
1   sword_name = "Big Blade"
2   sword_damage = 6000
3   sword_weight = 2.23606797749979
4   sword_level = 3
5   message = "{name:s} is a sword that does {damage:_} damage, weighs {weight:3}kg and requires level {level:<3}."
6   print(message.format(name=sword_name, damage=sword_damage, weight=sword_weight, level=sword_level))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_19.py
Big Blade is a sword that does 6_000 damage, weighs 2.23606797749979kg and requires level 3  .
PS C:\Users\hazeldotzone\Python Code>
```

The `:s` means "convert it as a string" (calls `str()` on it for us) this is the default and we don't need to specify

Field Conversions

lesson007_20.py

```
1   tebibyte = 1024**4
2   print(f"One tebibyte is {tebibyte} bytes.")
3   print(f"One tebibyte is {tebibyte:d} bytes.")
4   print(f"One tebibyte is {tebibyte:,} bytes.")
5   print(f"One tebibyte is {tebibyte:b} bytes.")
6   print(f"One tebibyte is {tebibyte:o} bytes.")
7   print(f"One tebibyte is {tebibyte:x} bytes.")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_20.py
One tebibyte is 1099511627776 bytes.
One tebibyte is 1099511627776 bytes.
One tebibyte is 1,099,511,627,776 bytes.
One tebibyte is 10000000000000000000000000000000000000000 bytes.
One tebibyte is 20000000000000 bytes.
One tebibyte is 10000000000 bytes.
PS C:\Users\hazeldotzone\Python Code>
```

decimal (the default)

with commas

binary

octal

hexadecimal

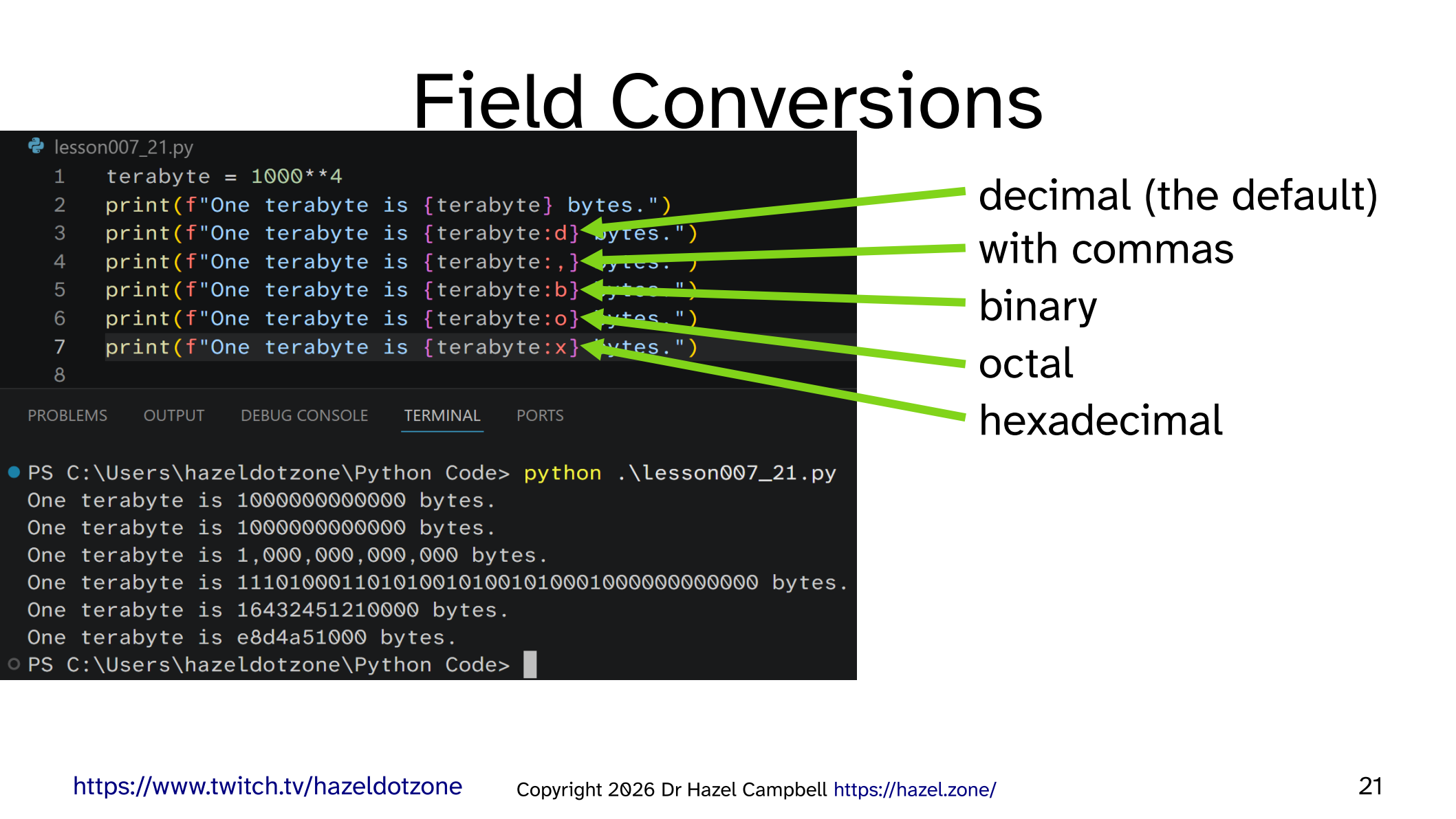
Field Conversions

lesson007_21.py

```
1   terabyte = 1000**4
2   print(f"One terabyte is {terabyte} bytes.")
3   print(f"One terabyte is {terabyte:d} bytes.")
4   print(f"One terabyte is {terabyte:,} bytes.")
5   print(f"One terabyte is {terabyte:b} bytes.")
6   print(f"One terabyte is {terabyte:o} bytes.")
7   print(f"One terabyte is {terabyte:x} bytes.")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_21.py
One terabyte is 1000000000000 bytes.
One terabyte is 1000000000000 bytes.
One terabyte is 1,000,000,000,000 bytes.
One terabyte is 11101000110101001010010100010000000000000 bytes.
One terabyte is 16432451210000 bytes.
One terabyte is e8d4a51000 bytes.
PS C:\Users\hazeldotzone\Python Code>
```



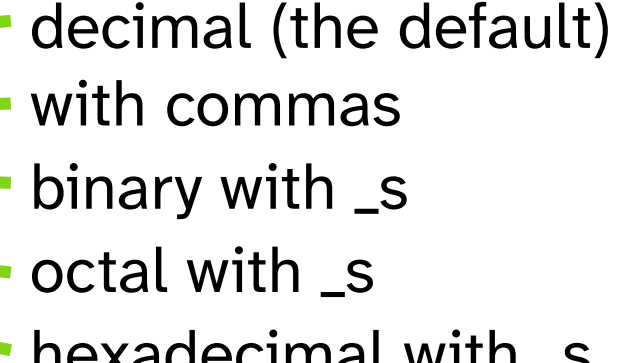
Field Conversions

lesson007_22.py

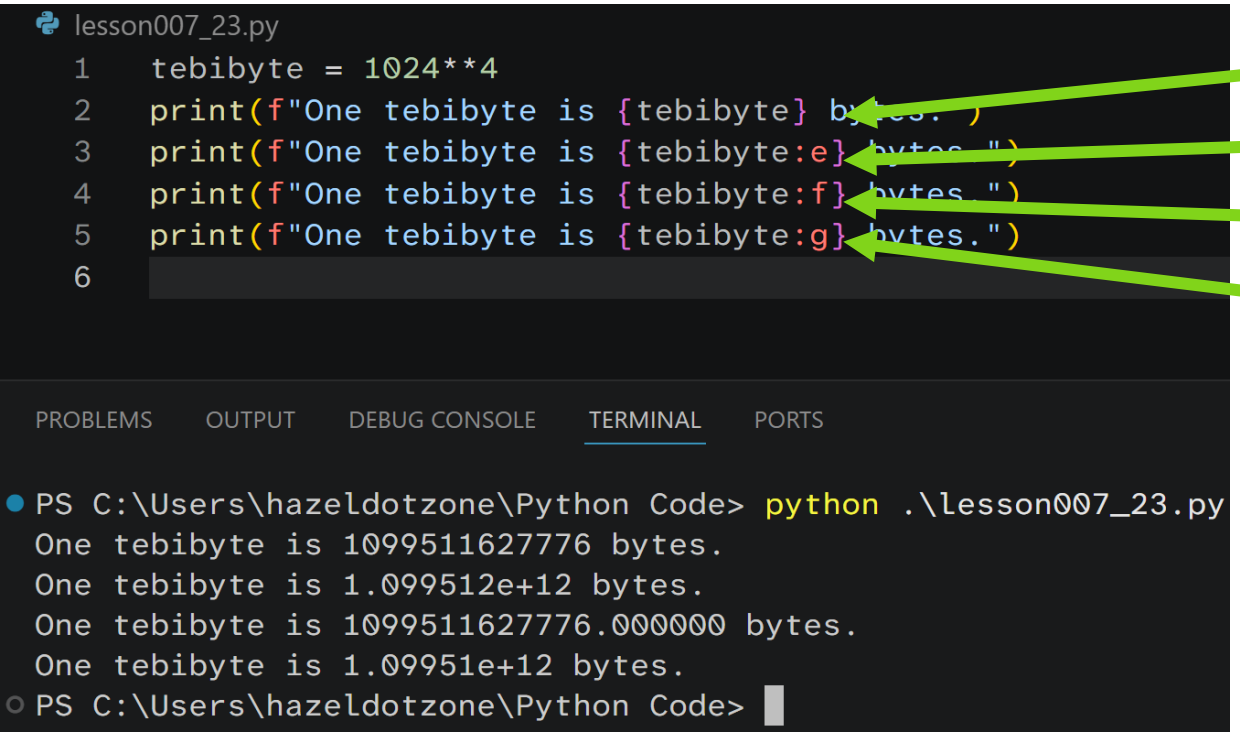
```
1   terabyte = 1000**4
2   print(f"One terabyte is {terabyte} bytes.")
3   print(f"One terabyte is {terabyte:d} bytes.")
4   print(f"One terabyte is {terabyte:,} bytes.")
5   print(f"One terabyte is {terabyte:_b} bytes.")
6   print(f"One terabyte is {terabyte:_o} bytes.")
7   print(f"One terabyte is {terabyte:_x} bytes.")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_22.py
One terabyte is 1000000000000 bytes.
One terabyte is 1000000000000 bytes.
One terabyte is 1,000,000,000,000 bytes.
One terabyte is 1110_1000_1101_0100_1010_0101_0001_0000_0000_0000 bytes.
One terabyte is 16_4324_5121_0000 bytes.
One terabyte is e8_d4a5_1000 bytes.
PS C:\Users\hazeldotzone\Python Code>
```



Field Conversions



decimal (the default)

scientific notation

use six decimals

maybe scientific notation

Field Conversions

lesson007_24.py

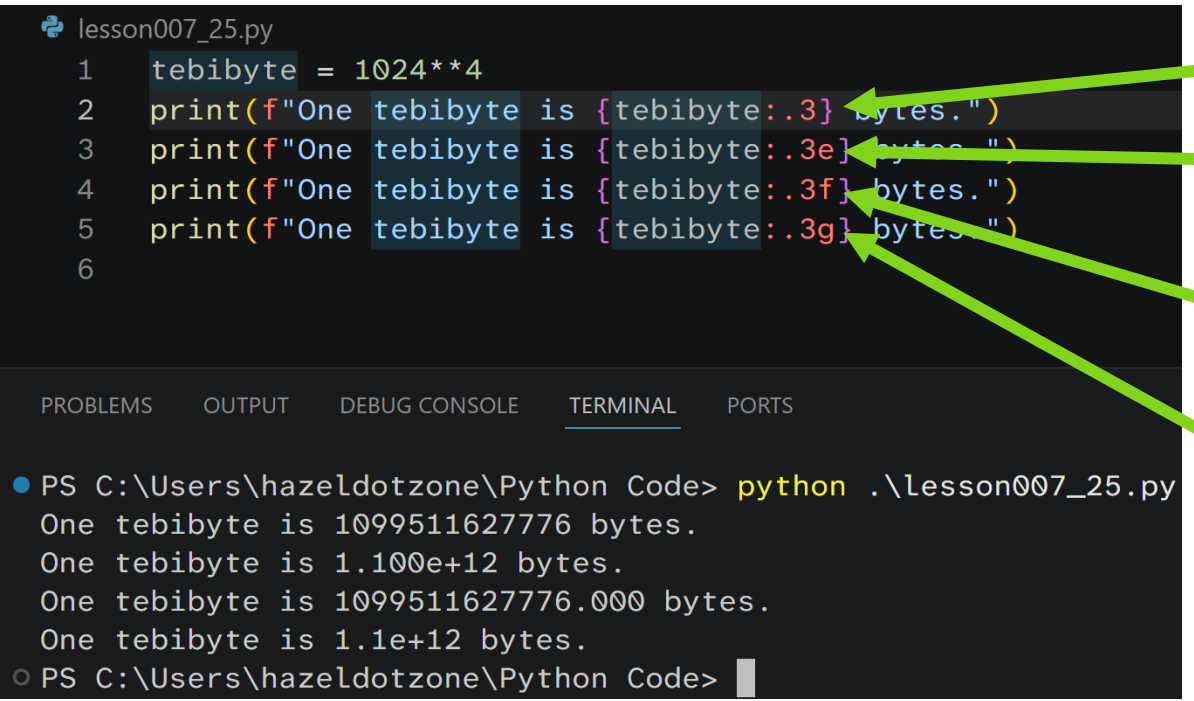
```
1   print(f"One kibibit is {1024/8:g} bytes.")
2   print(f"One kibibyte is {1024:g} bytes.")
3   print(f"One mebibit is {1024**2/8:g} bytes.")
4   print(f"One mebibyte is {1024**2:g} bytes.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_24.py
One kibibit is 128 bytes.
One kibibyte is 1024 bytes.
One mebibit is 131072 bytes.
One mebibyte is 1.04858e+06 bytes.
PS C:\Users\hazeldotzone\Python Code>
```

:g switches to scientific notation when the number gets big

Field Precision



decimal (the default)

scientific notation,
3 decimal places

3 decimal places

maybe scientific notation,
3 sig figs

Field Precision

lesson007_26.py

```
1   tebibyte = 1024**4
2   terabyte = 1000**4
3   ratio = terabyte/tebibyte
4   print(f"One terabyte is {ratio:.3} of a tebibyte.")
5   print(f"One terabyte is {ratio:.3e} of a tebibyte.")
6   print(f"One terabyte is {ratio:.3f} of a tebibyte.")
7   print(f"One terabyte is {ratio:.3g} of a tebibyte.")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_26.py
One terabyte is 0.909 of a tebibyte.
One terabyte is 9.095e-01 of a tebibyte.
One terabyte is 0.909 of a tebibyte.
One terabyte is 0.909 of a tebibyte.
PS C:\Users\hazeldotzone\Python Code>
```

3 decimal places

scientific notation,
3 decimal places

3 decimal places

maybe scientific notation,
3 sig figs

Field Precision & Width

lesson007_27.py

```
1   tebibyte = 1024**4
2   terabyte = 1000**4
3   ratio = terabyte/tebibyte
4   print(f"One terabyte is {ratio:9.3} of a tebibyte.")
5   print(f"One terabyte is {ratio:9.3e} of a tebibyte.")
6   print(f"One terabyte is {ratio:9.3f} of a tebibyte.")
7   print(f"One terabyte is {ratio:9.3g} of a tebibyte.")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_27.py
One terabyte is     0.909 of a tebibyte.
One terabyte is 9.095e-01 of a tebibyte.
One terabyte is     0.909 of a tebibyte.
One terabyte is     0.909 of a tebibyte.
PS C:\Users\hazeldotzone\Python Code>
```

3 decimal places

scientific notation,
3 decimal places

3 decimal places

maybe scientific notation,
3 sig figs

Field Precision & Width

lesson007_28.py

```
1   tebibyte = 1024**4
2   terabyte = 1000**4
3   ratio = terabyte/tebibyte
4   print(f"One terabyte is {ratio:09.3} of a tebibyte.")
5   print(f"One terabyte is {ratio:09.3e} of a tebibyte.")
6   print(f"One terabyte is {ratio:09.3f} of a tebibyte.")
7   print(f"One terabyte is {ratio:09.3g} of a tebibyte.")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_28.py
One terabyte is 00000.909 of a tebibyte.
One terabyte is 9.095e-01 of a tebibyte.
One terabyte is 00000.909 of a tebibyte.
One terabyte is 00000.909 of a tebibyte.
PS C:\Users\hazeldotzone\Python Code>
```

3 decimal places

scientific notation,
3 decimal places

3 decimal places

maybe scientific notation,
3 sig figs

Make the leading space 0s

Field Precision & Width

lesson007_29.py

```
1   tebibyte = 1024**4
2   terabyte = 1000**4
3   ratio = terabyte/tebibyte
4   print(f"One terabyte is {ratio:+10.3} of a tebibyte.")
5   print(f"One terabyte is {ratio:+10.3e} of a tebibyte.")
6   print(f"One terabyte is {ratio:+10.3f} of a tebibyte.")
7   print(f"One terabyte is {ratio:+10.3g} of a tebibyte.")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_29.py
One terabyte is     +0.909 of a tebibyte.
One terabyte is +9.095e-01 of a tebibyte.
One terabyte is     +0.909 of a tebibyte.
One terabyte is     +0.909 of a tebibyte.
PS C:\Users\hazeldotzone\Python Code>
```

3 decimal places

scientific notation,
3 decimal places

3 decimal places

maybe scientific notation,
3 sig figs

Show the sign

Field Precision & Width

lesson007_30.py

```
1   tebibyte = 1024**4
2   terabyte = 1000**4
3   ratio = terabyte/tebibyte
4   print(f"One terabyte is {ratio:+010.3} of a tebibyte.")
5   print(f"One terabyte is {ratio:+010.3e} of a tebibyte.")
6   print(f"One terabyte is {ratio:+010.3f} of a tebibyte.")
7   print(f"One terabyte is {ratio:+010.3g} of a tebibyte.")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_30.py
One terabyte is +00000.909 of a tebibyte.
One terabyte is +9.095e-01 of a tebibyte.
One terabyte is +00000.909 of a tebibyte.
One terabyte is +00000.909 of a tebibyte.
PS C:\Users\hazeldotzone\Python Code>
```

3 decimal places

scientific notation,
3 decimal places

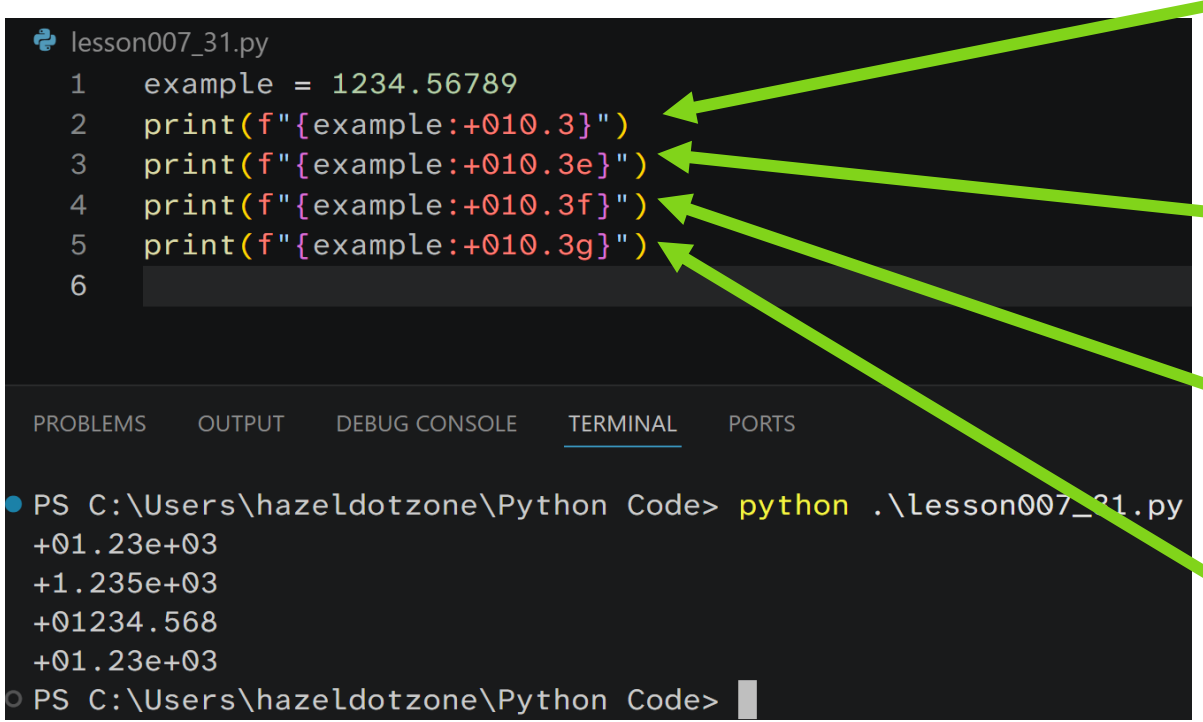
3 decimal places

maybe scientific notation,
3 sig figs

Show the sign

Make the leading space 0s

Field Precision & Width



3 decimal places

scientific notation,
3 decimal places

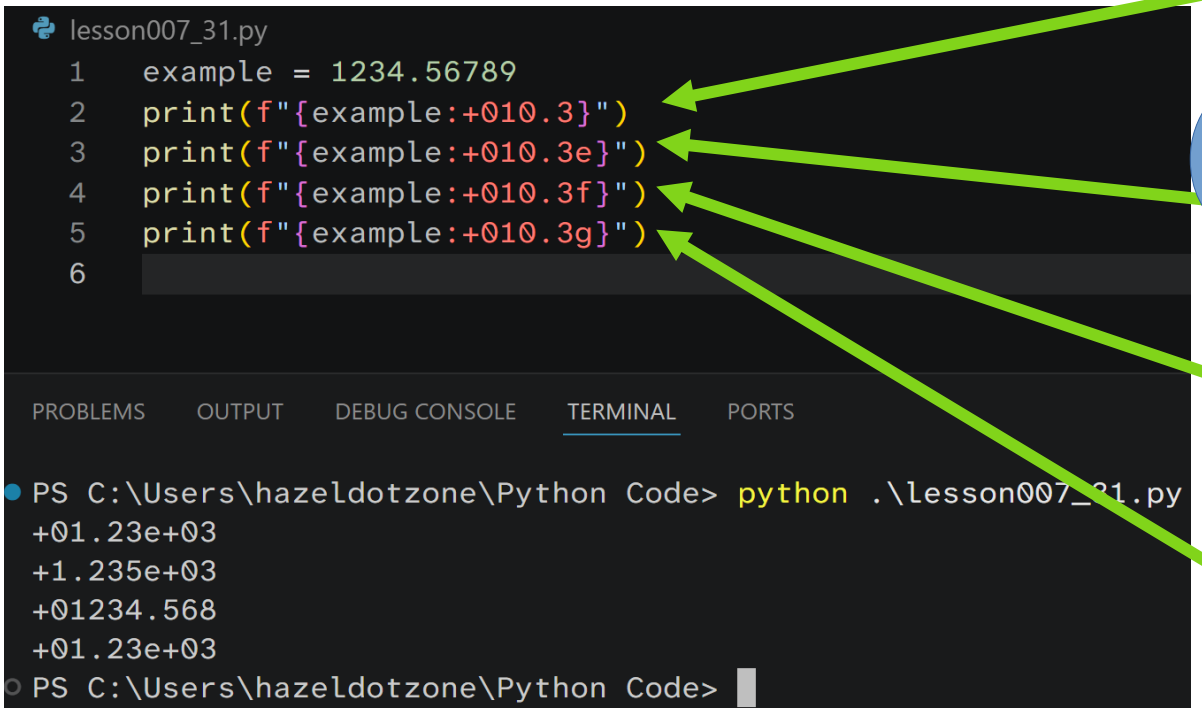
3 decimal places

maybe scientific notation,
3 sig figs

Show the sign

Make the leading space 0s

Field Precision & Width



3

in general, use g (that's what it stands for, g for general)

3

maybe scientific notation, 3 sig figs

Show the sign

Make the leading space 0s

Field Precision & Width

lesson007_33.py

```
1   tebibyte = 1024**4
2   terabyte = 1000**4
3   ratio = terabyte/tebibyte
4   print(f"One terabyte is {ratio:.3} of a tebibyte.")
5   print(f"One terabyte is {ratio:.3e} of a tebibyte.")
6   print(f"One terabyte is {ratio:.3f} of a tebibyte.")
7   print(f"One terabyte is {ratio:.3g} of a tebibyte.")
8   print(f"One terabyte is {ratio:.3%} of a tebibyte.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_33.py
One terabyte is 0.909 of a tebibyte.
One terabyte is 9.095e-01 of a tebibyte.
One terabyte is 0.909 of a tebibyte.
One terabyte is 0.909 of a tebibyte.
One terabyte is 90.949% of a tebibyte.
PS C:\Users\hazeldotzone\Python Code>
```

3 decimal places

scientific notation,
3 decimal places

3 decimal places

maybe scientific notation,
3 sig figs

hey, % is actually doing percentages!

Field Precision & Width & Conversions, etc.

No! They can be found in the Python reference.

https://docs.python.org/3/library/string.html#format-examples

Using Number Formatting

lesson007_35.py

```
1   miles = float(input("Enter a number of miles: "))
2   fmt = "{miles:12.4g} miles is {amount:12.4g} {unit}"
3   print(fmt.format(miles=miles, amount=(miles * 1.609344), unit="km"))
4   print(fmt.format(miles=miles, amount=(miles * 1760.0), unit="yards"))
5   print(fmt.format(miles=miles, amount=(miles * 5280.0), unit="feet"))
6   print(fmt.format(miles=miles, amount=(miles * 0.869), unit="nautical miles"))
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_35.py
Enter a number of miles: 3
          3 miles is        4.828 km
          3 miles is         5280 yards
          3 miles is    1.584e+04 feet
          3 miles is        2.607 nautical miles
PS C:\Users\hazeldotzone\Python Code>
```

String Conversions

lesson007_34.py

```
1   example_float = 2**0.5 # square root of 2, the first number to ever
2   example_int = 987543210
3   example_str = "hello!"
4   print(f"with !s: {example_float!s} ... with !r: {example_float!r}")
5   print(f"with !s: {example_int!s} ... with !r: {example_int!r}")
6   print(f"with !s: {example_str!s} ... with !r: {example_str!r}")
7
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_34.py
with !s: 1.4142135623730951 ... with !r: 1.4142135623730951
with !s: 987543210 ... with !r: 987543210
with !s: hello! ... with !r: 'hello!'
PS C:\Users\hazeldotzone\Python Code>
```

!s provides the usual string conversion

!r provides the "programmer's representation"

repr() and !r

```
PS C:\Users\hazeldotzone\Python Code> python
Python 3.12.12 (main, Oct 10 2025, 12:42:57)  [GCC UCRT 15.
Type "help", "copyright", "credits" or "license" for more i
>>> repr
<built-in function repr>
>>> print(str('She said "y\'all" in a southern drawl.'))
She said "y'all" in a southern drawl.
>>> print(repr('She said "y\'all" in a southern drawl.'))
'She said "y\'all" in a southern drawl.'
>>> print(f'{'She said "y\'all" in a southern drawl.'!r}')
'She said "y\'all" in a southern drawl.'
>>>
```

`repr()` and `!r`

!r is just a short way to get repr in your format strings.

repr() provides the programmer's representation.

For strings, this means it will show you the delimiters and escape sequences.

repr() and !r

```
>>> print(str("Hazel wondered, \"what do y'all think python will do to represent thi
s string?\""))
Hazel wondered, "what do y'all think python will do to represent this string?"
>>> print(repr("Hazel wondered, \"what do y'all think python will do to represent th
is string?\""))
'Hazel wondered, "what do y\'all think python will do to represent this string?"
'
```

For strings, this means it will show you the delimiters and escape sequences...

Maybe not the ones you put in, but they will represent the same string.

`repr()` and the REPL

```
>>> print(str("Hazel wondered, \"what do y'all think python will do to represent thi
s string?\""))
Hazel wondered, "what do y'all think python will do to represent this string?"
>>> print(repr("Hazel wondered, \"what do y'all think python will do to represent th
is string?\""))
'Hazel wondered, "what do y\'all think python will do to represent this string?"
'
```

For strings, this means it will show you the delimiters and escape sequences...

Maybe not the ones you put in, but they will represent the same string.

`repr()` and the REPL

```
>>> "a"+'b'
'ab'
```

The Python REPL actually uses repr() to print out the result of an r-evaluation.

Unless of course, the r-expression you gave it evaluated to None.

`repr()` and the REPL

```
>>> print(repr("a'b"+'c"d'))
'a\'bc"d'
>>> "a'b"+'c"d'
'a\'bc"d'
```

The Python REPL actually uses repr() to print out the result of an r-evaluation.

Unless of course, the r-expression you gave it evaluated to None.

`repr()` and the REPL

```
>>> "a normal string"
'a normal string'
>>> repr("a normal string")
"'a normal string'"
```

Do not put repr() into the REPL without something else... will double repr(repr())

Using repr()

lesson007_43.py

```
1   number = 11
2   print(number, type(number))
3   number_repr = repr(number)
4   print(number_repr, type(repr(number_repr)))
5
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python .\lesson007_43.py
11 <class 'int'>
11 <class 'str'>
PS C:\Users\hazeldotzone\Python Code>
```

Like str(), repr() converts things to str

Using repr()

```
>>> number = 11
>>> str(11) == repr(11)
True
```

```
>>> string = "twitch.tv/hazeldotzone"
>>> str(string) == repr(string)
False
```

For numbers, it's the same result

For strings, it's NOT the same result

Using repr()

```
>>> string = "twitch.tv/hazeldotzone"
>>> str(string) == string
True
```

When you call str with a str you just get back the same str

Using repr()

```
>>> string = "twitch.tv/hazeldotzone"
>>> str(string) == string
True
```

When you call str with a str you just get back the same str

Using repr()

- When you call repr() with a string you get a string containing Python code to recreate the string
- Not the same
  - Delimiters
  - Escape sequences

```
>>> print(string)
twitch.tv/hazeldotzone
>>> print(str(string))
twitch.tv/hazeldotzone
>>> print(repr(string))
'twitch.tv/hazeldotzone'
```

Using repr()

- When you call repr() with a string you get a string containing Python code to recreate the string
- Not the same
  - Delimiters
  - Escape sequences

```
>>> print(string, len(string))
twitch.tv/hazeldotzone 22
>>> print(str(string), len(str(string)))
twitch.tv/hazeldotzone 22
>>> print(repr(string), len(repr(string)))
'twitch.tv/hazeldotzone' 24
```

Using repr()

- When you call repr() with a string you get a string containing Python code to recreate the string
- Not the same code you used to construct the string, Python doesn't remember that. But code that will create the same string.

```
>>> quotes = '"'+"'"
>>> print(quotes)
"'
>>> print(repr(quotes))
'"\''
>>> quotes == '"\''
True
```

The Programmable Electronic Digital Computer

1) Represent everything as a sequence of numbers
2) Take a program
3) Take some input
4) Get warm for a while
5) Produce some output

The Programmable Electronic Digital Computer

1) Represent everything as a sequence of numbers
2) Take a program (represented as a sequence of numbers)
3) Take some input (represented as a sequence of numbers)
4) Get warm for a while
5) Produce some output (represented as a sequence of numbers)

ord() and chr()

- Take a character and provide its numeric code
- Take a number and convert it to the character it encodes

ord() and chr()

- In Python, a character is just a string of length 1

```
>>> ord('a')
97
>>> ord('aa')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: ord() expected a character, but string of length 2 found
```

ord() and chr()

lesson007_53.py

```
1   character = input("Enter a character: ")
2   number = ord(character)
3   print(f"The character {character!r} is coded by the number {number:07} (hex {number:06x})")
4   next_number = number + 1
5   next_character = chr(next_number)
6   print(f"The next character is {next_character!r}")
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson007_53.py
Enter a character: a
The character 'a' is coded by the number 0000097 (hex 000061)
The next character is 'b'
PS C:\Users\hazeldotzone\Python Code>
```

ord() and chr()

use repr

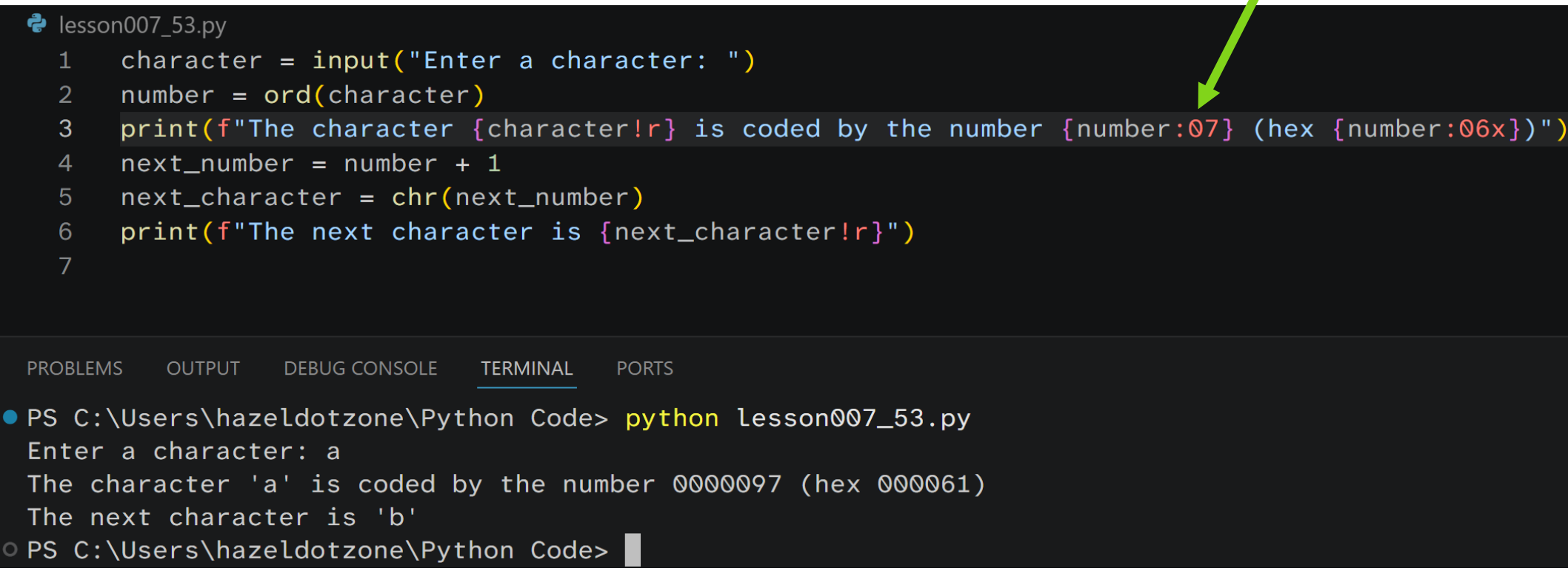
lesson007_53.py

```
1  character = input("Enter a character: ")
2  number = ord(character)
3  print(f"The character {character!r} is coded by the number {number:07} (hex {number:06x})")
4  next_number = number + 1
5  next_character = chr(next_number)
6  print(f"The next character is {next_character!r}")
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

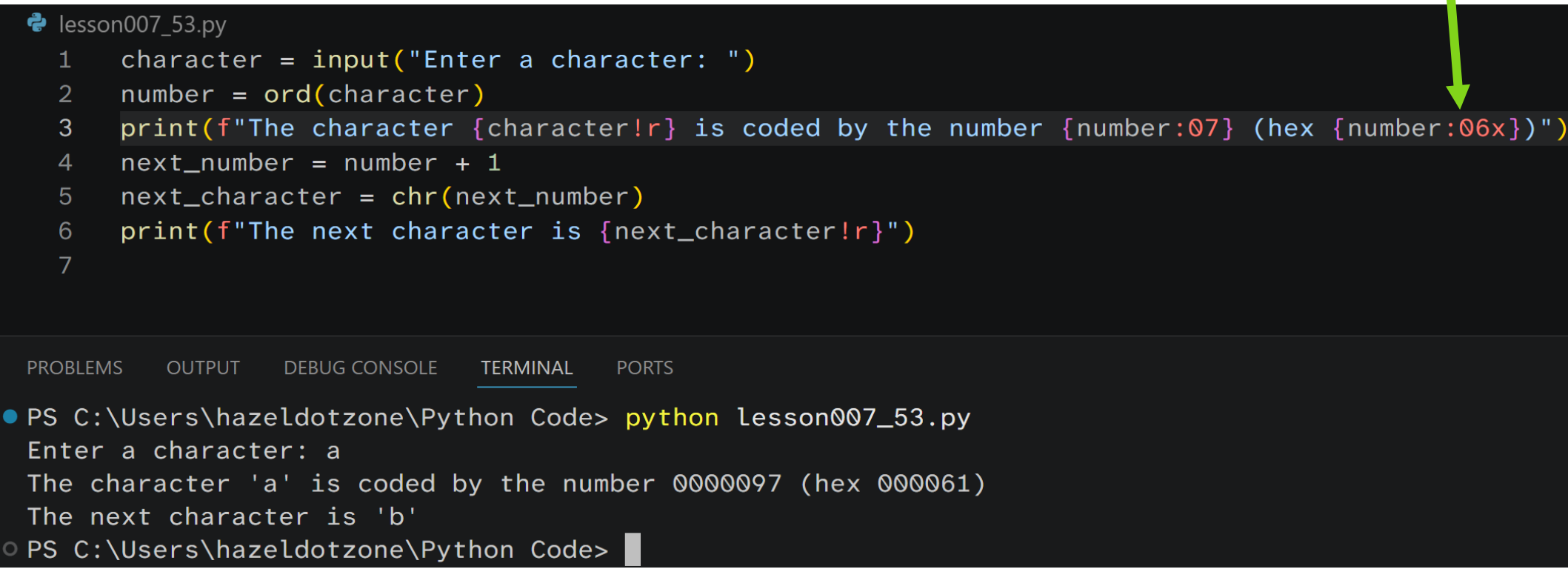
```
PS C:\Users\hazeldotzone\Python Code> python lesson007_53.py
Enter a character: a
The character 'a' is coded by the number 0000097 (hex 000061)
The next character is 'b'
PS C:\Users\hazeldotzone\Python Code>
```

ord() and chr()



ord() and chr()

6 hexadecimal digits



ord() and chr()

6 hexadecimal digits

lesson007_53.py

```
1   character = input("Enter a character: ")
2   number = ord(character)
3   print(f"The character {character!r} is coded by the number {number:07} (hex {number:06x})")
4   next_number = number + 1
5   next_character = chr(next_number)
6   print(f"The next character is {next_character!r}")
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson007_53.py
Enter a character: a
The character 'a' is coded by the number 0000097 (hex 000061)
The next character is 'b'
PS C:\Users\hazeldotzone\Python Code>
```

The Unicode Encoding

- It's what Python uses, and most things.
- Introduced in 1991
- Based on ISO-8859-1, from 1985
- Based on ASCII, from 1963 used for teleprinters (teletypes)
- Based on ITA2 (1932)
- Based on Murray code (1901) Baudot code (1874)

The Unicode Encoding

- It's what Python uses, and most things.
- Introduced in 1991
- Sometimes other encodings are used, mostly in Asia or older software.

The Unicode Encoding

- It's just a mapping of numbers to characters. What character is what number, what number is what character.
- That's it.
- That's how computers work.

The Unicode Encoding

- Emojis:

lesson007_53.py

```
1  character = input("Enter a character: ")
2  number = ord(character)
3  print(f"The character {character!r} is coded by the number {number:07} (hex {number:06x})")
4  next_number = number + 1
5  next_character = chr(next_number)
6  print(f"The next character is {next_character!r}")
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hazeldotzone\Python Code> python lesson007_53.py
Enter a character: 😊
The character '😊' is coded by the number 0128522 (hex 01f60a)
The next character is '😋'
PS C:\Users\hazeldotzone\Python Code>
```