1. Use equal sign to print variable name together with the value

```
likes = 1000000
print(f"{likes=}")
# >>> likes=1000000
```

2. Add colon and comma to make the output easier to read

```
print(f"{likes=:,}")
# >>> likes=1,000,000
```

3. Add colon .Nf to round-up the number in the output (where N is a natural number)

```
likes = 1000000.2654
print(f"{likes=:.1f}")
# >>> likes=1000000.3
```

# 4. Combine multiple f-strings with parentheses for cleaner formatting

```
obj = "cookie"
long_str = (
    f"This is my first {obj}.\n"
    f"This is my second {obj}.\n"
    f"This is my third {obj}."
)

print(long_str)
# >>> This is my first cookie.
# >>> This is my second cookie.
# >>> This is my third cookie.
```

5. Say you want to print a variable and its value but you don't want to see the equal sign. The output of f-string is evaluated to a simple string, so you can edit it as you would a string variable

```
likes = 1000000
print(f"{likes=:,}."
      .replace("=","" is equal to "")
      .capitalize()
)
# >>> Likes is equal to 1,000,000.
```

## 6. You can use f-string to access variable name

```
some_variable_name = (
f"{likes=}"[:f"{likes=}".rfind("=")]
)
print(some_variable_name)
# >>> 'likes'
```

7. Use it in __init__ together with `exec` for more precise constructor (probably not the best idea :D)

```
class A:
    def __init__(self, like, cookie, comment):
        prefix = 'self.'
        exec(f"{prefix}{like=}")
        exec(f"{prefix}{cookie=}")
        exec(f"{prefix}{comment=}")
    def __str__(self):
        return str(self.__dict__).replace(",
","\n").replace("'","")[1:-1]
like, cookie, comment = 100000, 20000, "I like
browser cookies"
print(A(like,cookie,comment))
# >>> like: 100000
# >>> cookie: 20000
# >>> comment: I like browser cookies
```

8. If we've gone this far, we might pass f-strings instead of arguments, and make the __init__ method a one liner

```
class A2:
    def __init__(self, *args):
        for arg in args: exec(f"self.{arg}")
    def __str__(self):
        return (str(self.__dict__)
                .replace(", ","\n")
                .replace("'","")[1:-1]
        )

print(A2(f"{like=}", f"{cookie=}",
f"{comment=}"))
# >>> like: 100000
# >>> cookie: 20000
# >>> comment: I like browser cookies
```

# 9. It even works for dictionaries!

```
other = {"what":42,"is_this":"afk"}
a2 = A2(
 f"{like=}",
 f"{cookie=}",
 f"{comment=}",
 f"{other=}"
)
print(a2)
# >>> like: 100000
# >>> cookie: 20000
# >>> comment: I like browser cookies
# >>> other: {what: 42
# >>> is_this: afk}
print(type(a2.other))
# >>> <class 'dict'>
```

## 10. A more conservative approach would be to use the logic from A2 class to generate the traditional look of the constructor:

```
class A3:
   def __init__(self, *args):
       for arg in args: exec(f"self.{arg}")
       arg_names = [arg[:arg.rfind("=")] for arg in args]
       print(f"def __init__(self, {', '.join(arg_names)}):")
       for arg in args:
           print(f"    self.{arg}".replace("="," = "))
   def __str__(self):
       return str(self.__dict__).replace(",
","\n").replace("'","")[1:-1]

a3 = A3(f"{like=}", f"{cookie=}", f"{comment=}", f"{other=}")
# >>> def __init__(self, like, cookie, comment, other):
# >>>     self.like = 100000
# >>>     self.cookie = 20000
# >>>     self.comment = 'I like browser cookies'
# >>>     self.other = {'what': 42, 'is_this': 'afk'}
```

**Although you'll probably not use the more creative tricks, it's still fun to play around with and see how Python actually works!**