code step by step bank account class:

```
class BankAccount:
    def __init__(self, name, balance = 0.00):
        self.name = name
        self.balance = balance

    def deposit(self, ammount):
        if ammount <= 0:
            return
        self.balance += ammount

    def withdraw(self, ammount):
        if ammount < 0:
            return
        if ammount > self.balance:
            return
        self.balance -= ammount
account = BankAccount("John", 100.00)
account.deposit(50.00)
account.withdraw(30.00)
print(account.name)
print(account.balance)
```

```
John
120.0
```

code step by step bank account class plus transaction fee:

```
class BankAccount:
    def __init__(self, name, balance = 0.00, transaction_fee = 0.00):
        self._name = name
        self._balance = balance
        self._transaction_fee = 0.00
        self.transaction_fee = transaction_fee

    @property
    def name(self):
        return self._name

    @property
    def balance(self):
        return self._balance

    @property
    def transaction_fee(self):
        return self._transaction_fee

    @transaction_fee.setter
```

```
    def transaction_fee(self, fee):
        if fee >= 0:
            self._transaction_fee = fee

    def deposit(self, amount):
        if amount > 0:
            self._balance += amount

    def withdraw(self, amount):
        if amount < 0:
            return
        withdraw_total = amount + self._transaction_fee
        if withdraw_total <= self._balance:
            self._balance -= withdraw_total

account = BankAccount("John", 100.00)
account.transaction_fee = 2.00
account.deposit(50.00)
account.withdraw(30.00)
print(account.name)
print(account.balance)
```

```
John
118.0
```

code step by step bank account str method:

```
class BankAccount:
    def __init__(self, name, balance = 0.00, transaction_fee = 0.00):
        self._name = name
        self._balance = balance
        self._transaction_fee = 0.00
        self.transaction_fee = transaction_fee

    @property
    def name(self):
        return self._name

    @property
    def balance(self):
        return self._balance

    @property
    def transaction_fee(self):
        return self._transaction_fee

    @transaction_fee.setter
    def transaction_fee(self, fee):
        if fee >= 0:
            self._transaction_fee = fee
```

```
    def deposit(self, amount):
        if amount > 0:
            self._balance += amount

    def withdraw(self, amount):
        if amount < 0:
            return
        withdraw_total = amount + self._transaction_fee
        if withdraw_total <= self._balance:
            self._balance -= withdraw_total

    def __str__(self):
        return f"{self.name}, ${self.balance:.2f}"

account = BankAccount("John", 100.00)
account.transaction_fee = 2.00
account.deposit(50.00)
account.withdraw(30.00)
account.__str__()
```

```
'John, $118.00'
```

code step by step factorial function:

```
def factorial(n):
    if n == 0:
        return 1
    if n == 1:
        return 1
    else:
        return(n * factorial(n - 1))

print(factorial(5))
```

```
120
```

data lemur problem:

```
factorial(5) # input

def factorial(n):
  x = 1 # base case
  for i in range(1, n+1): # go from 1 to n
    x *= i # update answer incrementally
  print(x)
  return x


120
```