Lab 7: Inheritance, Linked Lists

lab07.zip (lab07.zip)

Due by 11:59pm on Wednesday, March 13.

Starter Files

Download lab07.zip (lab07.zip). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the Ok (ok) autograder.

Required Questions

Getting Started Videos

Inheritance

Consult the drop-down if you need a refresher on Inheritance. It's okay to skip directly to the questions and refer back here should you get stuck. Inheritance

Q1: WWPD: Inheritance ABCs

Important: For all WWPD questions, type `Function` if you believe the answer is `<function...>`, `Error` if it errors, and `Nothing` if nothing is displayed.

Use Ok to test your knowledge with the following "What Would Python Display?" questions:

```
python3 ok -q inheritance-abc -u
```

```
>>> class A:
...   x, y = 0, 0
...   def __init__(self):
...         return
>>> class B(A):
...   def __init__(self):
...         return
>>> class C(A):
...   def __init__(self):
...         return
>>> print(A.x, B.x, C.x)
______

>>> B.x = 2
>>> print(A.x, B.x, C.x)
______

>>> A.x += 1
>>> print(A.x, B.x, C.x)
______

>>> obj = C()
>>> obj.y = 1
>>> C.y == obj.y
______

>>> A.y = obj.y
>>> print(A.y, B.y, C.y, obj.y)
______
```

Class Practice

Let's say we'd like to model a bank account that can handle interactions such as depositing funds or gaining interest on current funds. In the following questions, we will be building off of the `Account` class. Here's our current definition of the class:

```
class Account:
    """An account has a balance and a holder.
    >>> a = Account('John')
    >>> a.deposit(10)
    10
    >>> a.balance
    10
    >>> a.interest
    0.02
    >>> a.time_to_retire(10.25)  # 10 -> 10.2 -> 10.404
    2
    >>> a.balance                # Calling time_to_retire method should not change the ba
    10
    >>> a.time_to_retire(11)     # 10 -> 10.2 -> ... -> 11.040808032
    5
    >>> a.time_to_retire(100)
    117
    """
    max_withdrawal = 10
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance

    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds"
        if amount > self.max_withdrawal:
            return "Can't withdraw that amount"
        self.balance = self.balance - amount
        return self.balance
```

Q2: Retirement

Add a `time_to_retire` method to the `Account` class. This method takes in an `amount` and returns how many years the holder would need to wait in order for the current `balance` to grow to at least `amount`, assuming that the bank adds the interest (calculated as the current `balance` multiplied by the `interest` rate) to the `balance` at the end of each year.

```
    def time_to_retire(self, amount):
        """Return the number of years until balance would grow to amount."""
        assert self.balance > 0 and amount > 0 and self.interest > 0
        "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q Account
```

Q3: FreeChecking

Implement the `FreeChecking` class, which is like the `Account` class from lecture except that it charges a withdraw fee `withdraw_fee` after withdrawing `free_withdrawals` number of times. If a withdrawal is unsuccessful, it still counts towards the number of free withdrawals remaining, but no fee for the withdrawal will be charged.

```
class FreeChecking(Account):
    """A bank account that charges for withdrawals, but the first two are free!
    >>> ch = FreeChecking('Jack')
    >>> ch.balance = 20
    >>> ch.withdraw(100)  # First one's free. Still counts as a free withdrawal even thoug
    'Insufficient funds'
    >>> ch.withdraw(3)    # Second withdrawal is also free
    17
    >>> ch.balance
    17
    >>> ch.withdraw(3)    # Ok, two free withdrawals is enough, as free_withdrawals is onl
    13
    >>> ch.withdraw(3)
    9
    >>> ch2 = FreeChecking('John')
    >>> ch2.balance = 10
    >>> ch2.withdraw(3) # No fee
    7
    >>> ch.withdraw(3)  # ch still charges a fee
    5
    >>> ch.withdraw(5)  # Not enough to cover fee + withdraw
    'Insufficient funds'
    """
    withdraw_fee = 1
    free_withdrawals = 2

    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q FreeChecking
```

Linked Lists

Consult the drop-down if you need a refresher on Linked Lists. It's okay to skip directly to the questions and refer back here should you get stuck.

Linked Lists

Q4: WWPD: Linked Lists

Read over the `Link` class. Make sure you understand the doctests.

Use Ok to test your knowledge with the following "What Would Python Display?" questions:

```
python3 ok -q link -u
```

Enter `Function` if you believe the answer is `<function ...>`, `Error` if it errors, and `Nothing` if nothing is displayed.

If you get stuck, try drawing out the box-and-pointer diagram for the linked list on a piece of paper or loading the `Link` class into the interpreter with `python3 -i lab08.py`.

```
>>> link = Link(1000)
>>> link.first
______

>>> link.rest is Link.empty
______

>>> link = Link(1000, 2000)
______

>>> link = Link(1000, Link())
______
```

```
>>> link = Link(1, Link(2, Link(3)))
>>> link.first
______

>>> link.rest.first
______

>>> link.rest.rest.rest is Link.empty
______

>>> link.first = 9001
>>> link.first
______

>>> link.rest = link.rest.rest
>>> link.rest.first
______

>>> link = Link(1)
>>> link.rest = link
>>> link.rest.rest is Link.empty
______

>>> link.rest.rest.rest.rest.first
______

>>> link = Link(2, Link(3, Link(4)))
>>> link2 = Link(1, link)
>>> link2.first
______

>>> link2.rest.first
______
```

```
>>> link = Link(5, Link(6, Link(7)))
>>> link                 # Look at the __repr__ method of Link
______

>>> print(link)          # Look at the __str__ method of Link
______
```

Q5: Duplicate Link

Write a function `duplicate_link` that takes in a linked list `s` and a value `val`. It mutates `s` so that each element equal to `val` is followed by an additional `val` (a duplicate copy). It returns `None`.

> **Note**: In order to insert a link into a linked list, reassign the `rest` attribute of the `Link` instances that have `val` as their `first`. Try drawing out a doctest to visualize!

```
def duplicate_link(s, val):
    """Mutates s so that each element equal to val is followed by another val.

    >>> x = Link(5, Link(4, Link(5)))
    >>> duplicate_link(x, 5)
    >>> x
    Link(5, Link(5, Link(4, Link(5, Link(5)))))
    >>> y = Link(2, Link(4, Link(6, Link(8))))
    >>> duplicate_link(y, 10)
    >>> y
    Link(2, Link(4, Link(6, Link(8))))
    >>> z = Link(1, Link(2, (Link(2, Link(3)))))
    >>> duplicate_link(z, 2) # ensures that back to back links with val are both duplicate
    >>> z
    Link(1, Link(2, Link(2, Link(2, Link(2, Link(3))))))
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q duplicate_link
```

Check Your Score Locally

You can locally check your score on each question of this assignment by running

```
python3 ok --score
```

This does NOT submit the assignment! When you are satisfied with your score, submit the assignment to Gradescope to receive credit for it.

Submit

Submit this assignment by uploading any files you've edited **to the appropriate Gradescope assignment.** Lab 00 (https://cs61a.org/lab/lab00/#submit-with-gradescope) has detailed instructions.

In addition, all students who are **not** in the mega lab must complete this attendance form (https://go.cs61a.org/lab-att). Submit this form each week, whether you attend lab or missed it for a good reason. The attendance form is not required for mega section students.