# Lab 7: Inheritance, Linked Lists

**lab07.zip (lab07.zip)**

*Due by 11:59pm on Wednesday, October 23.*

## Starter Files

Download lab07.zip (lab07.zip).

# 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 improve the `Account` class from lecture, which models a bank account that can process deposits and withdrawals.

```
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 the number of years until the current `balance` grows 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. Make sure you're not modifying the account's balance!

> **Important**: Calling the `time_to_retire` method should not change the account balance.

```
    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 except that it charges a withdraw fee `withdraw_fee` after withdrawing `free_withdrawals` number of times. If a withdrawal is unsuccessful, no withdrawal fee will be charged, but it still counts towards the number of free withdrawals remaining.

```
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)    # Now there is a fee because free_withdrawals is only 2
    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: Without One

Implement `without`, which takes a linked list `s` and a non-negative integer `i`. It returns a new linked list with all of the elements of `s` except the one at index `i`. (Assume `s.first` is the element at index 0.) The original linked list `s` should not be changed.

> **Hint:** Using recursive approach might be easier than the iterative approach.

```
def without(s, i):
    """Return a new linked list like s but without the element at index i.

    >>> s = Link(3, Link(5, Link(7, Link(9))))
    >>> without(s, 0)
    Link(5, Link(7, Link(9)))
    >>> without(s, 2)
    Link(3, Link(5, Link(9)))
    >>> without(s, 4)           # There is no index 4, so all of s is retained.
    Link(3, Link(5, Link(7, Link(9))))
    >>> without(s, 4) is not s  # Make sure a copy is created
    True
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q without
```

## Q6: 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`. Be careful not to get into an infinite loop where you keep duplicating the new copies!

> **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 Assignment

If you are in a regular section of CS 61A, fill out this lab attendance and feedback form (https://forms.gle/dHxj8gttNWRY6Ptm9). (If you are in the mega section, you don't need to fill out the form.)

Then, submit this assignment by uploading any files you've edited **to the appropriate Gradescope assignment.** Lab 00 (../lab00/#submit-with-gradescope) has detailed instructions.