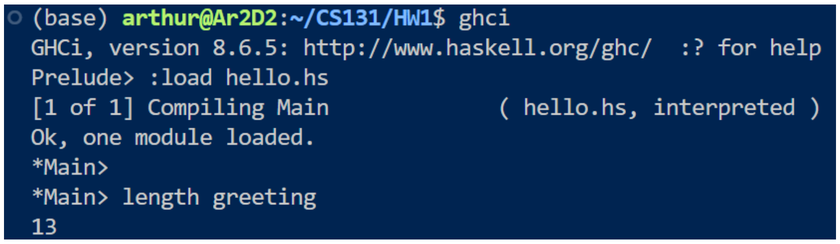
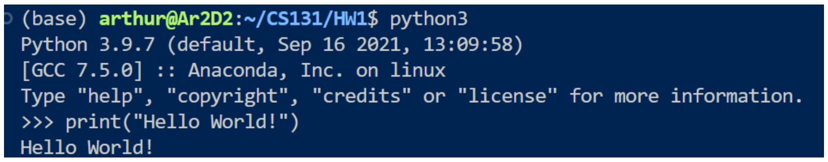
1. **INTR1**: Haskell Setup

The command is:
`length <string_name>`
where string_name is `greeting`
in this case.



```
(base) arthur@Ar2D2:~/CS131/HW1$ ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> :load hello.hs
[1 of 1] Compiling Main             ( hello.hs, interpreted )
Ok, one module loaded.
*Main>
*Main> length greeting
13
```

2. **INTR2**: Python Setup



```
(base) arthur@Ar2D2:~/CS131/HW1$ python3
Python 3.9.7 (default, Sep 16 2021, 13:09:58)
[GCC 7.5.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World!")
Hello World!
```

3. **PYTH1**: Object Reference Semantics, Parameter Passing, Shallow Copying, Boxing

Part A:

The reason why `num` doesn't change is because Python is ALWAYS pass by object/pointer. When we run `num *= 5` it is the equivalent of `num = num * 5` and when we assign `num` a new object not change the original `num` passed. For objects the values in the object are mutable and the objects point to the new mutated value.

Part B:

i:

```
joke6
joke3
joke4
```

While tracing through the code, we can see that the first two lines of the `process` function are fine. The third line will set the c pointer to `com + [Comedian("joke5")]` but the line after still alters the original `com` since the c pointer still does point to it. Also the line `c1 = Comedian("joke7")` will not alter anything since the pointers in the array point to the original object.

ii:

```
joke6
joke2
hello world
```

Since it is a shallow copy nothing sticks (you can think of it as passing by value). The only reason why joke1 got changed to joke6 was because we used the custom changer in the function that will always successfully change.

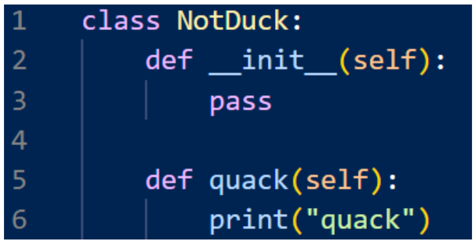
4. **PYTH2**: Duck Typing, Dunder Functions

Because Python follows the Duck typing ideology where everything is compiled at runtime. The motto is if it looks like a duck, quacks like a duck, then it is a duck. So Python doesn't actually care about the type of object when it comes to Foo and string, since they both have a length implementation. But int doesn't have one so that is why it fails.

5. **PYTH3**: Duck Typing, Easier To Ask For Forgiveness, Inheritance

Part A:

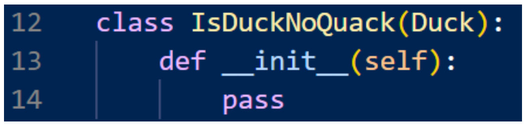
The reason why this works is because we first check if the NotDuck class has a quack() function. It does so it'll return true for is_duck_a(). On the other hand for is_duck_b() it compares NotDuck to Duck and since NotDuck isn't derived from Duck it'll return false.



```python
class NotDuck:
    def __init__(self):
        pass

    def quack(self):
        print("quack")
```

Part B:

This works since we don't have a quack() method so is_duck_a() would fail but IsDuckNoQuack is still Duck technically since it is derived from duck but is unable to quack.



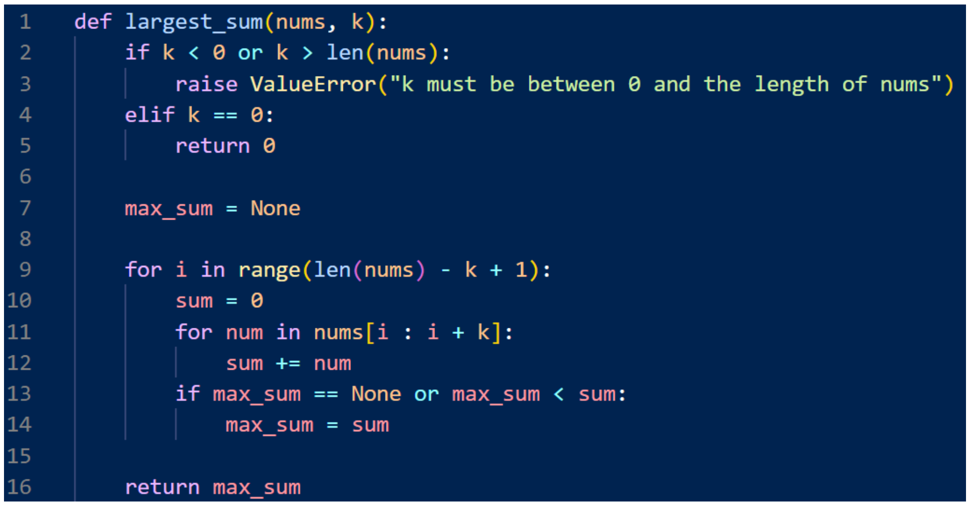
```python
class IsDuckNoQuack(Duck):
    def __init__(self):
        pass
```

Part C:

I would say A because Python is one of those languages that do things on the fly (everything being compiled at runtime) and if a duck is able to quack it could be a person imitating to be a duck and Python honestly wouldn't care. So it follows more of Python's ideology.

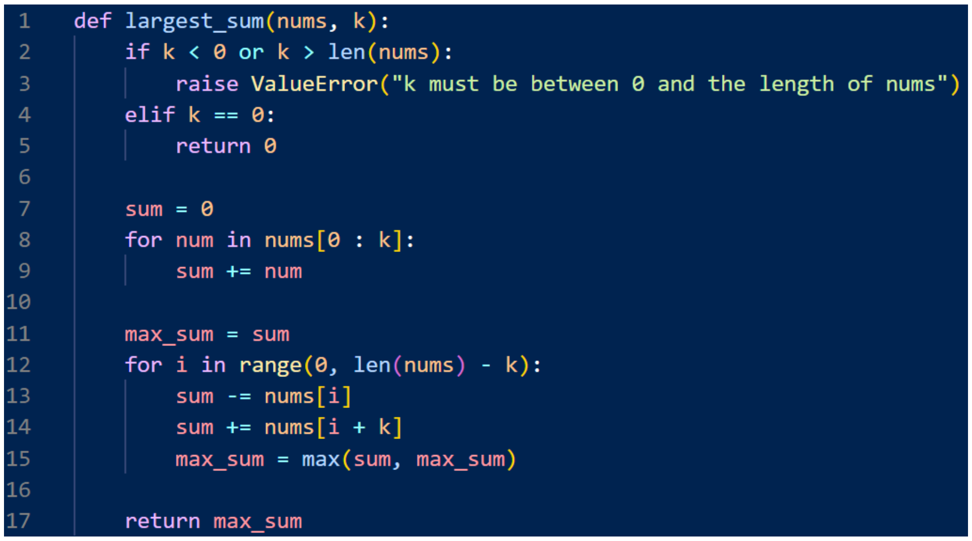
6. **PYTH4**: Slicing

Part A:



```python
def largest_sum(nums, k):
    if k < 0 or k > len(nums):
        raise ValueError("k must be between 0 and the length of nums")
    elif k == 0:
        return 0

    max_sum = None

    for i in range(len(nums) - k + 1):
        sum = 0
        for num in nums[i : i + k]:
            sum += num
        if max_sum == None or max_sum < sum:
            max_sum = sum

    return max_sum
```

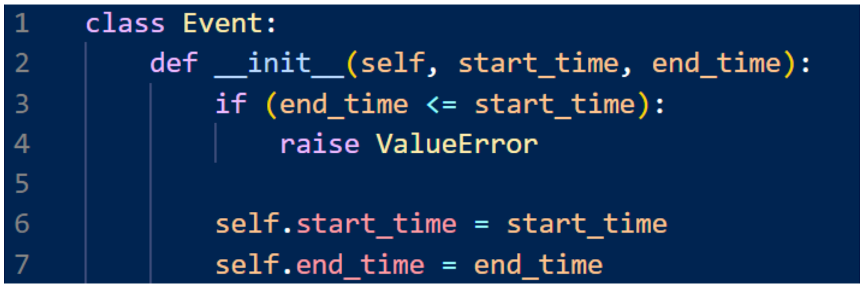
Part B:



```python
def largest_sum(nums, k):
    if k < 0 or k > len(nums):
        raise ValueError("k must be between 0 and the length of nums")
    elif k == 0:
        return 0

    sum = 0
    for num in nums[0 : k]:
        sum += num

    max_sum = sum
    for i in range(0, len(nums) - k):
        sum -= nums[i]
        sum += nums[i + k]
        max_sum = max(sum, max_sum)

    return max_sum
```

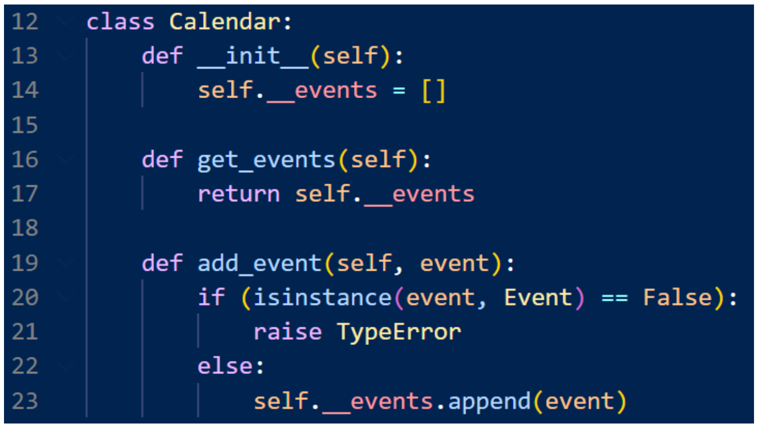
7. **PYTH5**: Inheritance, Exceptions

Part A:



```python
class Event:
    def __init__(self, start_time, end_time):
        if (end_time <= start_time):
            raise ValueError

        self.start_time = start_time
        self.end_time = end_time
```

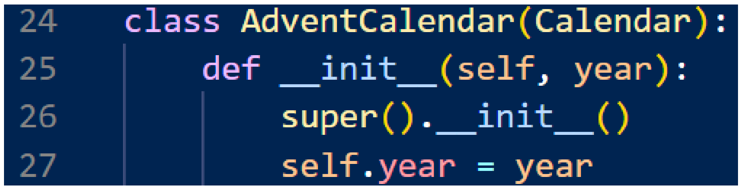
Part B:



```python
class Calendar:
    def __init__(self):
        self.__events = []

    def get_events(self):
        return self.__events

    def add_event(self, event):
        if (isinstance(event, Event) == False):
            raise TypeError
        else:
            self.__events.append(event)
```

Part C:

This is because AdventCalendar doesn't necessarily inherit __events from Calendar and when AdventCalendar tries to run



```python
class AdventCalendar(Calendar):
    def __init__(self, year):
        super().__init__()
        self.year = year
```

get_events() it'll try to look for _AdventCalendar.__events which doesn't exist. We can fix this by adding super().__init__() under AdventCalendar

8. **PYTH6**: Interpreted vs Compiled Languages

Part A:

C-lang should generally be faster since compiled languages have the entire program compiled into machine code in one go and it is easier for the machine to read and execute since the entire program is right there. On the other hand interpreted languages are generally slower because the compiler reads the code line by line and has to translate each line into machine language which adds the cost of extra overhead.

Part B:

Jamie is far likelier to have her server running first because she is using an interpreted language that usually comes with a ton of built-in libraries involving hosting websites on servers (hence why Python is used for so many webdev tools). She doesn't need to compile; she can just write her code and immediately execute it allowing for faster result and compilation. This also makes troubleshooting easier since you have faster development cycles.

On the other hand if you use a compiled language you have to compile the entire program after every time you changed it meaning that it'll add extra overhead in this scenario.

Part C:

Connie can read Jamie's script since Jamie's script is written in an interpreted language, and Connie has the appropriate interpreter installed on her SmackBook Pro, she can run the script without any issues. However it will be unable to execute Timmy's pre-compiled executable since Tim compiled his code specifically for Intel architecture, the executable is likely using an instruction set that is not compatible with the N1 chip. Without an emulator or a translation layer (like Rosetta), there is no way for Connie's computer to interpret the Intel-based executable. The proprietary nature of the N1's instruction set means that it cannot directly run code compiled for a different architecture.

9. **PYTH7**: Numpy

NumPy arrays are stored in consecutive blocks of memory, which is more cache-friendly and therefore faster than a manually implemented matrix multiplication that may involve more complex memory access patterns. Additionally managing multiple smaller data types of Python objects is reduced leading to better memory and reduced garbage collection overhead.

10. **PYTH8**: Class vs Instance Variables

Part A:

```
j.joke = I dressed as a UDP packet at the party. Nobody got it.
Joker.joke = I dressed as a UDP packet at the party. Nobody got it.
```

```
self.joke = I dressed as a UDP packet at the party. Nobody got it.
Joker.joke = I dressed as a UDP packet at the party. Nobody got it.
self.joke = Why do Java coders wear glasses? They can't C#.
Joker.joke = How does an OOP coder get wealthy? Inheritance.
j.joke = Why do Java coders wear glasses? They can't C#.
Joker.joke = How does an OOP coder get wealthy? Inheritance.
```
It does this since the first four lines have no change; we are just printing the same joke over again. The fifth and sixth lines involve changing the static variable as well as the member variable and since objects have legit changing it changes.

Part B:

Each statement prints out the corresponding text. The first four don't change and in the function we specifically altered both the member variable `joke` as well as the static class variable `joke` which are actually very different things!