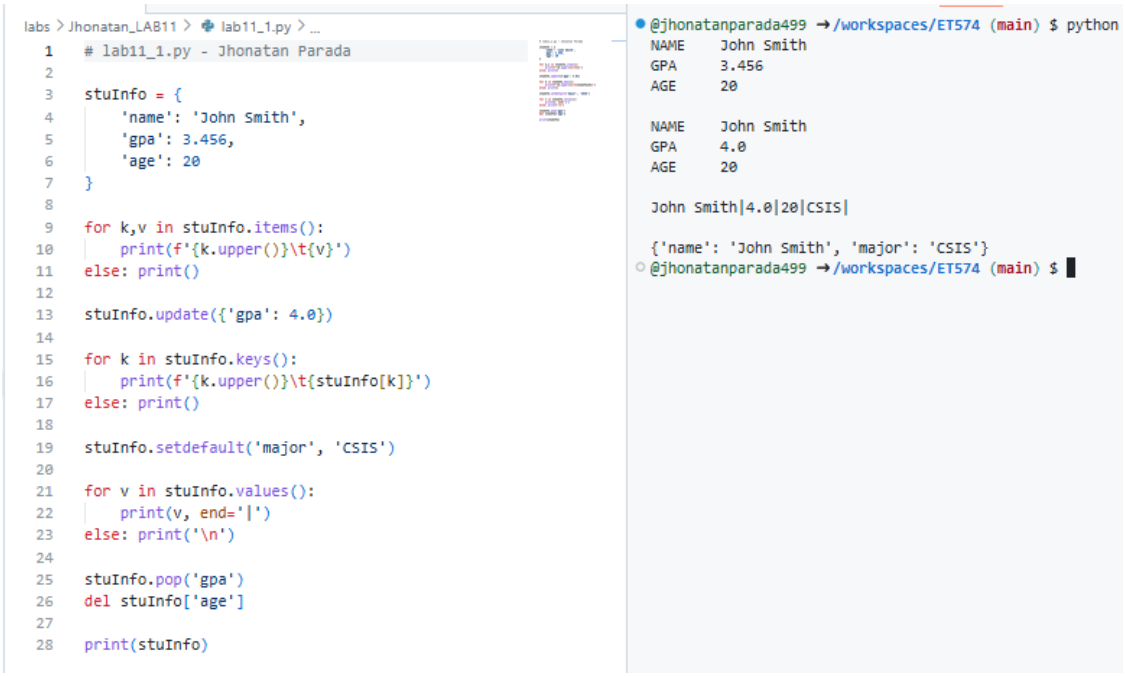
Lab 11 summary

Jhonatan Parada

ET574

1.



labs > Jhonatan_LAB11 > lab11_1.py > ...

```python
# lab11_1.py - Jhonatan Parada

stuInfo = {
    'name': 'John Smith',
    'gpa': 3.456,
    'age': 20
}

for k,v in stuInfo.items():
    print(f'{k.upper()}\t{v}')
else: print()

stuInfo.update({'gpa': 4.0})

for k in stuInfo.keys():
    print(f'{k.upper()}\t{stuInfo[k]}')
else: print()

stuInfo.setdefault('major', 'CSIS')

for v in stuInfo.values():
    print(v, end='|')
else: print('\n')

stuInfo.pop('gpa')
del stuInfo['age']

print(stuInfo)
```

```
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python
NAME    John Smith
GPA     3.456
AGE     20

NAME    John Smith
GPA     4.0
AGE     20

John Smith|4.0|20|CSIS|

{'name': 'John Smith', 'major': 'CSIS'}
@jhonatanparada499 ➜ /workspaces/ET574 (main) $
```

labs > Jhonatan_LAB11 > lab11_2.py > ...

```python
# lab11_2.py - Jhonatan Parada

rank = {
    1:"Freshman",
    2:"Sophmore",
    3:"Junior",
    4:"Senior"
}

years = input(
    f'Enter the # of years in the school <1-{len(rank)}>: '
)

try:
    years = int(years)
except:
    print('Invalid input')

if isinstance(years, int):
    if 1 < years > len(rank):
        print('Invalid years.')
    else:
        print(f'Year {years} = {rank[years]}')
```

```
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python
Enter the # of years in the school <1-4>: 1
Year 1 = Freshman
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python
Enter the # of years in the school <1-4>: 4
Year 4 = Senior
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python
Enter the # of years in the school <1-4>: 3
Year 3 = Junior
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python
Enter the # of years in the school <1-4>: abc
Invalid input
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python
Enter the # of years in the school <1-4>: 44
Invalid years.
@jhonatanparada499 ➜ /workspaces/ET574 (main) $
```

labs > Jhonatan_LAB11 > lab11_3a.py > ...

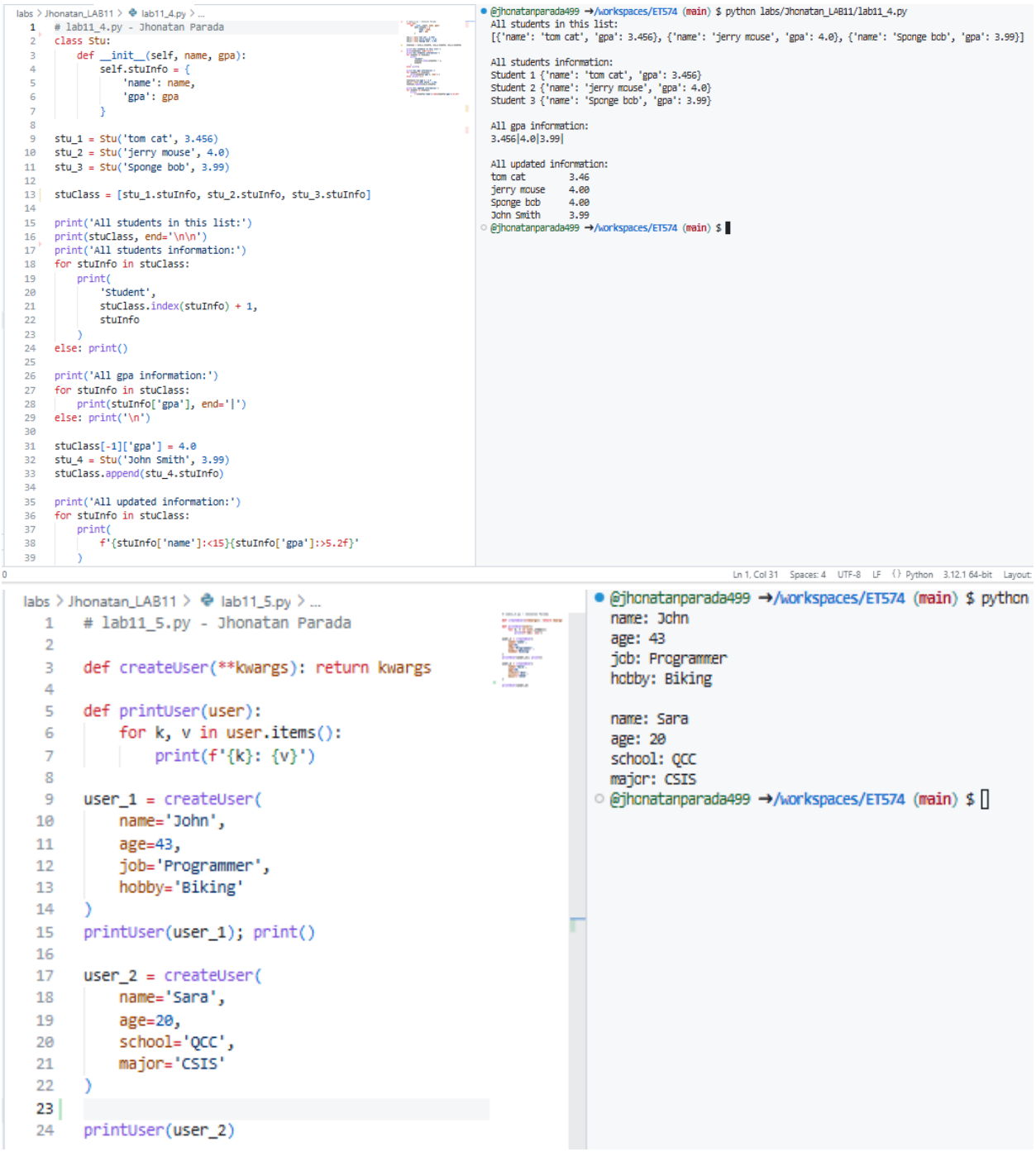
```python
# lab11_3a.py - Jhonatan Parada

import string

chars = [c for c in string.ascii_lowercase]
nums = [n for n in range(1, 26 + 1)]

charNum = dict(zip(chars, nums))

for k,v in charNum.items():
    print(k,v, end='|')
else: print()
```

```
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python labs/Jhonatan_LAB11/lab11_3a.py
a 1|b 2|c 3|d 4|e 5|f 6|g 7|h 8|i 9|j 10|k 11|l 12|m 13|n 14|o 15|p 16|q 17|r 18|s 19|t 20|u 21|v 22|w 23|x
 24|y 25|z 26|
@jhonatanparada499 ➜ /workspaces/ET574 (main) $
```

labs > Jhonatan_LAB11 > lab11_3b.py > ...

```python
# lab11_3b.py - Jhonatan Parada

import string
from lab11_3a import charNum

chars = [
    c for c in string.ascii_uppercase
]

nums = [
    n for n in range(100, 2600 + 100, 100)
]

numChar = dict(zip(nums, chars))

for k,v in numChar.items():
    print(k,v,end='|')
else: print()

charNum.update(numChar)

print(charNum)
```

```
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python labs/Jhonatan_LAB11/lab11_3b.py
a 1|b 2|c 3|d 4|e 5|f 6|g 7|h 8|i 9|j 10|k 11|l 12|m 13|n 14|o 15|p 16|q 17|r 18|s 19|t 20|u 21|v 22|w 23|x
 24|y 25|z 26|
100 A|200 B|300 C|400 D|500 E|600 F|700 G|800 H|900 I|1000 J|1100 K|1200 L|1300 M|1400 N|1500 O|1600 P|1700
 Q|1800 R|1900 S|2000 T|2100 U|2200 V|2300 W|2400 X|2500 Y|2600 Z|
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13
, 'n': 14, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20, 'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 2
5, 'z': 26, 100: 'A', 200: 'B', 300: 'C', 400: 'D', 500: 'E', 600: 'F', 700: 'G', 800: 'H', 900: 'I', 1000:
 'J', 1100: 'K', 1200: 'L', 1300: 'M', 1400: 'N', 1500: 'O', 1600: 'P', 1700: 'Q', 1800: 'R', 1900: 'S', 20
00: 'T', 2100: 'U', 2200: 'V', 2300: 'W', 2400: 'X', 2500: 'Y', 2600: 'Z'}
@jhonatanparada499 ➜ /workspaces/ET574 (main) $
```



2. I made a mistake in lab11_2.py in regards of the order of execution of the code. This happened because I usually prefer not to put code that I know it will not return error inside a try statement, in other words, I only put code that might generate errors inside a try statement, but I forgot to add an if statement afterwards to verify that the variable years was of type integer, without handling this exception, the code would function partially.

b. I had a conflict understanding [**kwargs] and [for k,v in dict.items()] at the beggining, but after the class of 11/26/2024, I came to the following conclusion:

**kwargs is a special parameter that accepts any number of arguments if the given arguments are written as of [variable=value,...]. It is interesting because at first I tried [variable:value] or [“variable”:value] since that is the syntax of the items of a dictionary, but that is not a valid syntax for an argument, so, the behavior ends up being something like using optional parameters when calling a function, therefore, I think one will not be able to create keys of type integer since this is not allowed to perform: [1=value] nor [‘1’=value], but we know that the keys of a dictionary do can be of type integer, I even wonder whether it is possible to write any key as a number no matter its data type.

Finally, the reason why there is two variables and the work fine in the syntax k,v when referring to dict.items() is because every item of the dictionary is going to be transformed into a tuple, therefore, that’s why they don't conflict with each other. In other words:

K, v = (a, b) (also equal to) k, v = a, b

Therefore:

For k,v in dict.items() (is equal to) for k,v in [(a, b),...]