A Hello User Program

- Decide on the purpose and composition of the project
  - Our project: Say hello to the user
  - Parameters firstName, middleName and lastName as parameters
  - No output, other than screen printing
  - Execution: Program greets user by full name
  - One function in project: main()

Setup Steps

- Make a working directory

```
mkdir project2
cd project2
```

- Use Poetry to create new project

```
poetry new hello_user
cd hello_user
```

- Add Project Dependencies

```
poetry add typer
poetry add rich
```

- Add Project Development Dependencies

```
poetry add -D black mypy
```

Mypy: http://mypy-lang.org/

Setup Steps

- Add File: project2/hello user/hello user/ init .py

```
"""Package-level docstring for hello_user package."""

__version__ = "0.1.0"
```

- Add File:projects/hello user/pyproject.toml

```
[project] ...

[tool.poetry.scripts]
hello_user = "hello_user.main:cli"

[tool.poetry.group.dev.dependencies] ...
```

- Update Poetry

```
poetry install
```

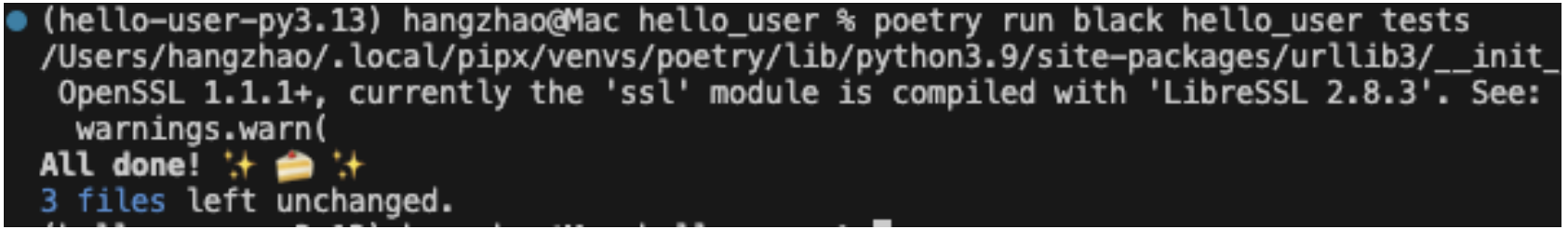
Add File: projects/hello user/hello user/main.py - File located in sanbox: main.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from rich.console import Console
import typer

# create a Typer object to support the command-line interface
cli = typer.Typer()
@cli.command()
def main(first: str = "", middle: str = "", last: str = ""):
                """Say hello to the person having a name of first, middle and last name"""
                console = Console()
                console.print(" Hello to;")
                console.print(f"\t First = {first}")
                console.print(f"\t Middle = {middle}")
                console.print(f"\t Last = {last}")
# end of main()
```

Basic Reformatting with Black

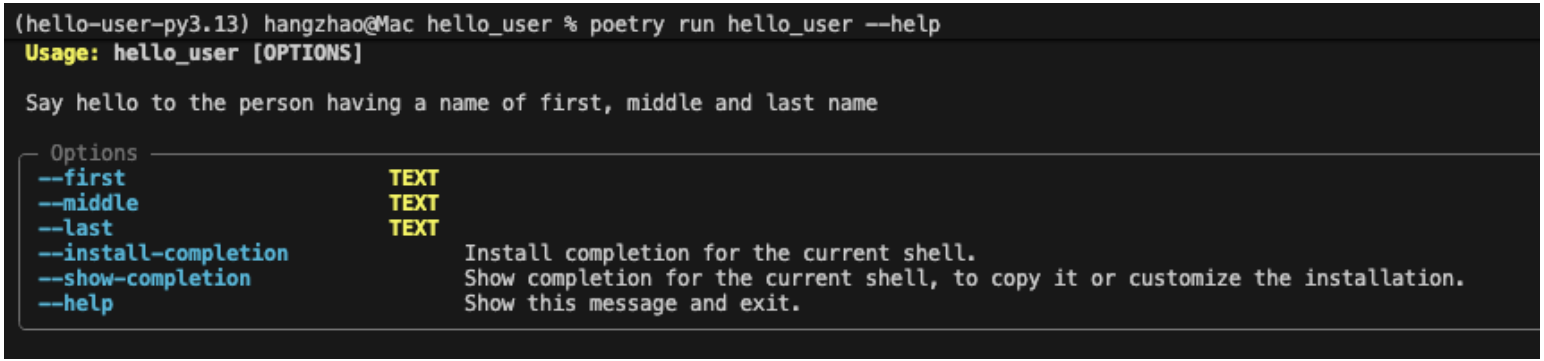
poetry run black hello_user tests



Execute Project

What do you see?

```
# run from projects/hello_user/hello_user
poetry run hello_user --help
```



Execute Project

- What do you see?

```
# run from projects/hello_user
poetry run hello_user
```

- Without parameters

```
poetry run hello_user
Hello to;
        first =
        middle =
        last =
```

Execute Project

- What do you see?

```
# run from projects/hello_user poetry run hello_user

--first John

--middle H.

--last Davis
```

- Without parameters

```
Hello to;
        first = John
        middle = H.
        last = Vader
```

Discrete Structures!

CMPSC 102

Data Containers

ALLEGHENY COLLEGE

Key Questions and Learning Objectives

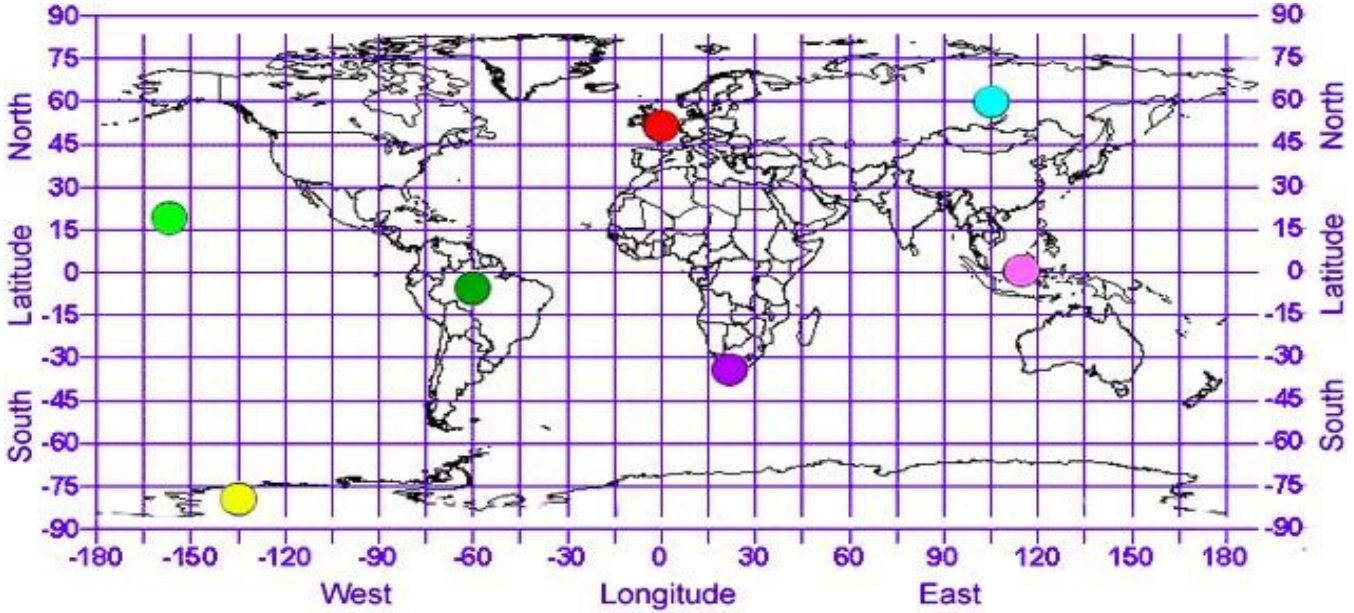
- How do I use the mathematical concepts of ordered pairs, n-tuples, lists and dictionaries to implement functions with a clearly specified behaviors?

- To remember and understand some discrete mathematics and Python programming concepts, enabling the investigation of practical applications

What are Ordered Pairs?

- Mathematical concepts yield predictable programs
- Understanding the concept of an ordered pair:
    - **Pair**: a grouping of two entities
    - **Ordered**: an order of entities matters
    - **Ordered Pair**: a grouping of two entities for which order matters
    - **Coordinate on Earth**: the latitude and longitude coordinates are an ordered pair
    - **Complex Numbers**: the real and imaginary parts are an ordered pair
    - An ordered pair is not the same as a set of two elements! Why?
    - Can we generalize to an ordered grouping beyond two entities? How?

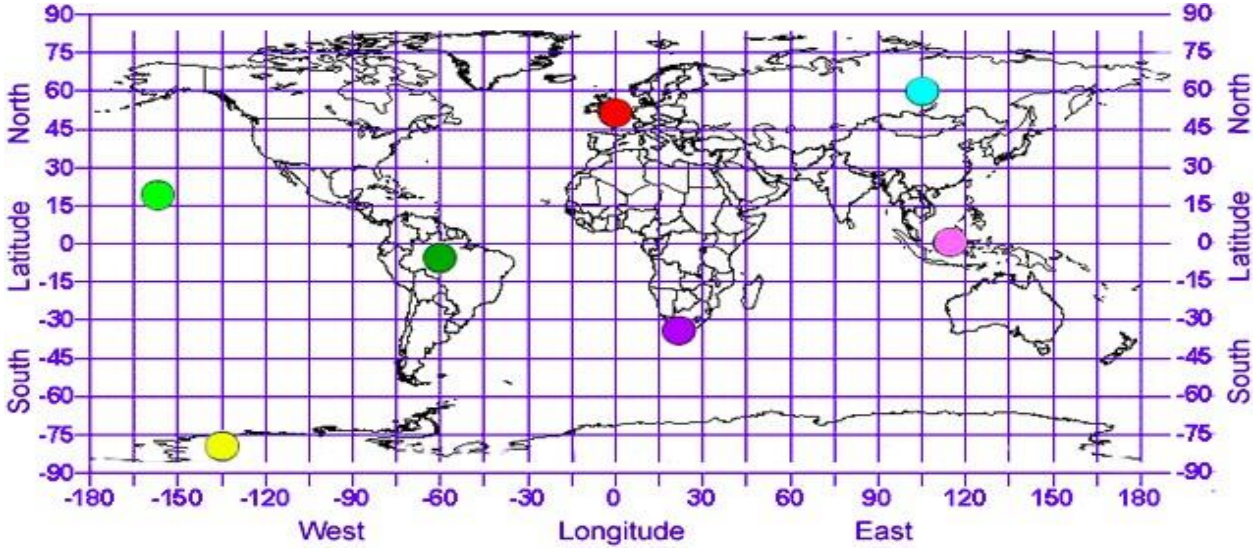
Practical Applications of Ordered Pairs



Ordered Pairs: A global address system

- Hawaii, USA 19.5429, 155.6659 (Green Dot)
- Paris, France (48.8566, 2.3522) (Red Dot)
- Meadville, PA: (41.6414, 80.1514)

Practical Applications of Ordered Pairs



Understanding the order of the pair

- Specified according to the standard (Latitude, Longitude)
- Why does the order matter for these pairs of location data?
- How do you interpret the positive and negative numbers?

Generalizing Ordered Pairs to n-Tuples

- We could have an "ordered triple" or "ordered quadruple"
- The *n*-tuple is the generic name for "tuples" of any size
  - A 2-tuple is the same as an **ordered pair**
  - A 3-tuple is the same as an **ordered triple**
  - A 4-tuple is the same as an **ordered quadruple**
  - n-tuples contain a **finite** number of entities
- We write *n*-tuples with notation like $(1, 2)$ or (x, y, z)
- Denoting *n*-tuples enable the **creation of new mathematical objects**
- While the type of entity in an *n*-tuple may be different, not every entity in the *n*-tuple must be different. This means that **duplicates are possible**!

Generalizing Ordered Pairs to n-Tuples


```
empty_tuple = ()
print(type(empty_tuple))

single_number = (3,)
print(type(single_number))
```

- Some tuples may not (yet) contain any data in them!
- Singleton tuples must use the comma notation

Generalizing Ordered Pairs to n-Tuples


```
what_var_a = (3)
print(type(what_var_a )) # What do you find?

what_var_b = (3,)
print(type(what_var_b )) # What do you find?

second_var = (3,4)
print(type(second_var)) # What do you find?
```

- What is the difference between a tuple and a number?

Tuples - A Tuple is a collection of Python objects separated by commas

- An empty tuple

```
empty_tuple = ()
print (empty_tuple)
print(type(empty_tuple)) # <class 'tuple'>
```

- A non-empty tuple

```
nonEmpty_tuple = ("a","b","c","d")
print(nonEmpty_tuple[0]) # 'a'
print(nonEmpty_tuple[len(nonEmpty_tuple)-1]) # gets last element: 'd'
```

- Check to see that elements are in a tuple

```
nonEmpty_tuple = ('a', 'b', 'c', 'd', 4, 'Hi')
print("Hi" in nonEmpty_tuple) # True
print(4 in nonEmpty_tuple) # True
print(3 in nonEmpty_tuple) # False
```

Tuples

- Checking for sub-elements in tuple

```
nonEmpty_tuple = ("a","b","c","d", 4, "Hi", "My music")
print(nonEmpty_tuple)

print("my" in nonEmpty_tuple) # False
print("My" in nonEmpty_tuple) # False
print("Hi" in nonEmpty_tuple) # True
print("HI" in nonEmpty_tuple) # False

# check to see if detail is in a substring in tuple
print("My" in nonEmpty_tuple[6]) # True
```

Adding to Tuples

- Convert tuple to list, add element, append, convert back

```
a_tuple = ('2',) #define Tuple
items = ['a', 'b', 'c', 'd'] # elements to add
l_list = list(a_tuple)# make a list

for x in items:
                l_list.append(x) # add items to list
#output as a tuple
print(tuple(l_list))
```

Adding and Removing items to Tuples

- combining two tuples

```
s_tuple = (1,2,3)
print(type(s_tuple)) # <class 'tuple'>
s_tuple = (1,2,3) + (3,4,5)
print(s_tuple) # (1, 2, 3, 3, 4, 5)
```

- tuple to list, remove element, list to tuple

```
s_tuple = (1,2,3)
print(type(s_tuple)) # <class 'tuple'>
s_tuple = list(s_tuple)
s_tuple .remove(1)
s_tuple = tuple(s_tuple)
print(s_tuple, type(s_tuple))
```

Iterating Through Elements in Tuples

- Iteration

```
nonEmpty_tuple = ("a","b","c","d", 4, "Hi", "My music")
for i in nonEmpty_tuple:
        print(i)
```

- Iteration

```
for i in range(len(nonEmpty_tuple)):
        print("i= ",i, "nonEmpty_tuple[i]=",nonEmpty_tuple[i])
```

- Note: With tuples (like lists), we know which element will be printed first (the first element, from above).

Packing and Unpacking Tuples

- Pack a tuple into a variable

```
pair = (3,4)
print(pair[0]) # 3
print(pair[1]) # 4
```

- Unpack the contents of a tuple

```
pair = (3,4)
x, y = pair
# (x, y) = pair
print(x) # Output: 3
print(y) # Output: 4
```

- Unpack and perform simultaneous assignment

```
x, y = 3, 4
x, y = y, x
print(x) # Output: 4
print(y) # Output: 3
```

Dictionaries - An array of a key and a value that is connected for quick searching

- A dictionary maps a set of objects (keys) to another set of objects (values).
- A Python dictionary is a mapping of unique keys to values.
- Dictionaries are mutable, which means they can be changed.
- The values that the keys point to can be any Python value

- An empty dictionary

```
myDictionary_dict = {}
print(myDictionary_dict)
print(type(myDictionary_dict)) # <class 'dict'>
```

Dictionaries

- Adding to a dictionary

```
myDictionary_dict = {}
myDictionary_dict[0] = "zero"
print(myDictionary_dict[0]) # gives 'zero'

myDictionary_dict[1] = "one"
print(myDictionary_dict) #{1: 'one', 0: 'zero'}
```

- Removing elements from a dictionary

```
myDictionary_dict = {}
myDictionary_dict[3] = "three"

del myDictionary_dict[3]
print(myDictionary_dict) #{} (is empty)
```

Randomly Choosing Elements

- Choosing Elements from a List

```
import random
abc_list = ['a','b','c','d','e']
print(random.choice(abc_list)) # 'c'
print(random.choice(abc_list)) # 'd'
```

- Choosing Elements from a List

```
import random
abc_set = set(['a','b','c','d','e']) # Convert list to a set
abc2_list = list(abc_set) # Convert set back to list
print(random.choice(abc2_list)) # 'd'
```

Randomly Choosing Elements

- Choosing Elements from a Dictionary

```
import random
abc_dict = {1:"one",2:"two",3:"Three"} # {keys: values}
num_list = list(abc_dict) # convert dict keys to list
n = random.choice(num_list) # pick a random key from the list
print(abc_dict[n]) # sub in n to get key value, two
```

Randomly Choosing A Number

- Choosing Elements from a Dictionary

```
import random

l_list = ['Joan', 'Jane', 'Jan', 'Janet']
my_friends = {
                "Joan": "814-555-1234",
                "Jane": "814-555-1235"
}
# print(f'{my_friends["Jane"]}')
name = random.choice(l_list)
number = my_friends[name]
print(f"Name: {name},\n number: {number}") # 'two'
```

Putting things together (1/3) - Functions to generate data

```
import random
## Generate some random data
# Function to generate a random phone number
def generate_phone_number():
        # (Note: return statement is on one line)
        return f"{random.randint(100, 999)}-{random.randint(100, 999)}-{random.randint(1000, 9999)}"
# end of generate_phone_number()

# Function to generate a random email address
def generate_email():
        domains = ["gmail.com", "yahoo.com", "hotmail.com", "example.com"]
        return f"{random.choice(domains)}"
# end of generate_email()
```

Putting things together (2/3) - Pair random data for dictionary

```
# List of random names_list
names_list = ["Alice", "Bob", "Charlie", "David", "Eve"]

# Creating the dictionary from which we select names_list
contacts = {}
for name in names_list:
        phone_number = generate_phone_number()

        # Note: email_address declaration all on one line
        email_address = f"{name.lower()}_{random.randint(1, 100)}@{generate_email()}"
        contacts[name] = [phone_number, email_address]

# Displaying the dictionary
print(f" My Contacts:\n {contacts}")
```

Putting things together (3/3) - Select a random name

```
# Randomly selecting a name
thisName = random.choice(names_list)
number = contacts[thisName][0]
email = contacts[thisName][1]

print("\n And the winner is ... \n")
print(f" Name: {thisName}")
print(f" number: {number}")
print(f" email: {email}")
```

Discrete Structures!

CMPSC 102

Data Containers

ALLEGHENY COLLEGE

Key Questions and Learning Objectives

- How do I use the mathematical concepts of ordered pairs, n-tuples, lists and dictionaries to implement functions with a clearly specified behaviors?

- To remember and understand some discrete mathematics and Python programming concepts, enabling the investigation of practical applications

Combining Dictionaries and Lists - create a list of data

```
# define Alice's list
detailsAlice=["555-8181", "Alice@...", "Paris"]
print(f" email: {detailsAlice[1]}")

# define Mike's list
detailsMike=["555-1234", "michael@...", "Meadville"]
print(f" email: {detailsMike[1]}")

# create dictionary
contacts = {}
#add details as key, value assignment
contacts["Alice"] = detailsAlice
contacts["Mike"] = detailsMike

for i in contacts: # extract details
                print(f"{i} -> {contacts[i]}")
```

More with Dictionaries and Lists - Part 1

```
multsOfTwo = []
for i in range(10):
        multsOfTwo.append(i**2)
print(f"multsOfTwo : {multsOfTwo}")
# multsOfTwo : [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

multsOfTwo = []
multsOfThree = []
multsOfFour = []
for i in range(10):
        multsOfTwo.append(i**2)
        multsOfThree.append(i**3)
        multsOfFour.append(i**4)

print(f"multsOfTwo : {multsOfTwo}") # : [0,1,4,...,81]
print(f"multsOfThree : {multsOfThree}") # : [0,1,8,...,729]
print(f"multsOfFour : {multsOfFour}") # : [0,1,16,...,6561]
```

More with Dictionaries and Lists - Part 2

- Add all lists to a dictionary

```
# assign dictionary
multiples = {}
multiples["twos"] = multsOfTwo
multiples["three"] = multsOfThree
multiples["four"] = multsOfFour

for i in multiples:
        print(f" multiples of {i} -> {multiples[i]}")
```

Data in the Form of Tuples

- Comma separate value (CSV) are frequently used in business and science!
- How can we input this file of n-tuples into a Python program?
- How do we parse each line based on a delimiter?
- How can the program handle multiple-word content with commas?

CSV Data - Files in Directories Can Store n-Tuples

- Suppose you had some data in a CSV format?
- How to do something with the data?!

- CSV data: sandbox/contacts.csv

```
tylernelson@gmail.com,Careers adviser
gregory02@medina-mayer.com,"Accountant, management"
jonesmiguel@hotmail.com,Health and safety inspector
rsanchez@yahoo.com,"Surveyor, planning and development"
hillfrank@ward-wood.com,"Scientist, physiological"
aaronhunter@gmail.com,"Surveyor, insurance"
kylebarnes@hotmail.com,Records manager
joe70@yahoo.com,Network engineer
torresjames@white.info,Electrical engineer
shawkins@watson.com,Science writer
```

Functions that Manage Tuples

File: csvreader.py

```
from os.path import exists
from logging import exception

def openCSVFile(fname_str: str) -> str:
        """loads a file, returns csv string"""
        # print("openCSVFile()")
        if not exists(fname_str): # no file found?
                print(f"\t [-] No file by that name: {fname_str}")
                exit() # end program if no file has been found.
        try:
                data_str = open(fname_str, "r").read()
        except exception:
                print("\t [-] Using correct filename?")
                return None
        # commas in this loaded file?
        if len(data_str) > 0 and "," in data_str:
                return data_str
        return None
```

Functions that Manage Tuples - iterateData

```
def iterateData(in_str: str) -> dict:
        """Function to output the data in tidy lines. Place data into dictionary."""
        contact_dict = {}
        for line in in_str.splitlines():
                # print(line)
                # get the name, located before first comma
                name_str = line[: line.find(",")]
                service_str = line[line.find(",") + 1 :].replace("'", "")
                contact_dict[name_str] = service_str
        return contact_dict
```

Functions that Manage Tuples - main()

```
def main() -> None:
        """driver function"""
        prompt_str = "\t Enter the CSV filename : "
        myFile_str = input(prompt_str)
        # print(f"\t [+] You entered file : {myFile_str}")

        myCSV_str = openCSVFile(myFile_str)
        # print(f"Main() {myCSV_str}")

        # print out in tidy lines
        myContact_dict = iterateData(myCSV_str)

        # print(f"Dictionary of names: {myContact_dict}")
        for i in myContact_dict:
                print(f"\t [+] {i} : {myContact_dict[i]}")
```

Fun Activity - Ungraded work

- Prepare Python code to implement the following code
  - Create a tuple with some data
  - Now, change the tuple into a list
  - Now, combine the list with a dictionary
  - Now, place a tuple in the dictionary
  - Print out the contents of each data structure