Discrete Structures: Data Containers CMPSC 102

Oliver BONHAM-CARTER

Spring 2024
Week 5
Slides 01

Let's Discuss

Key Questions

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

Learning Objectives

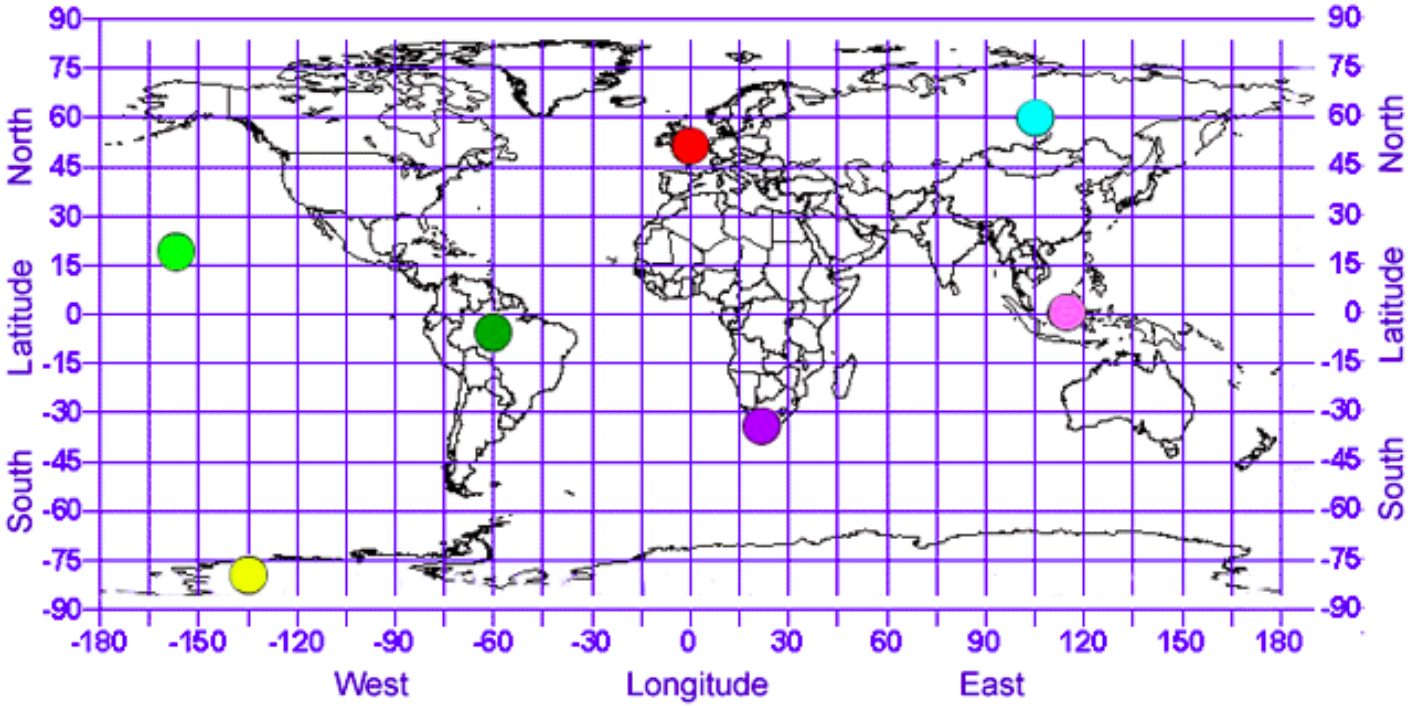
To **remember** and **understand** some discrete mathematics and Python programming concepts, enabling the investigation of practical applications

What are *Ordered Pairs*?

Some definitions

- 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



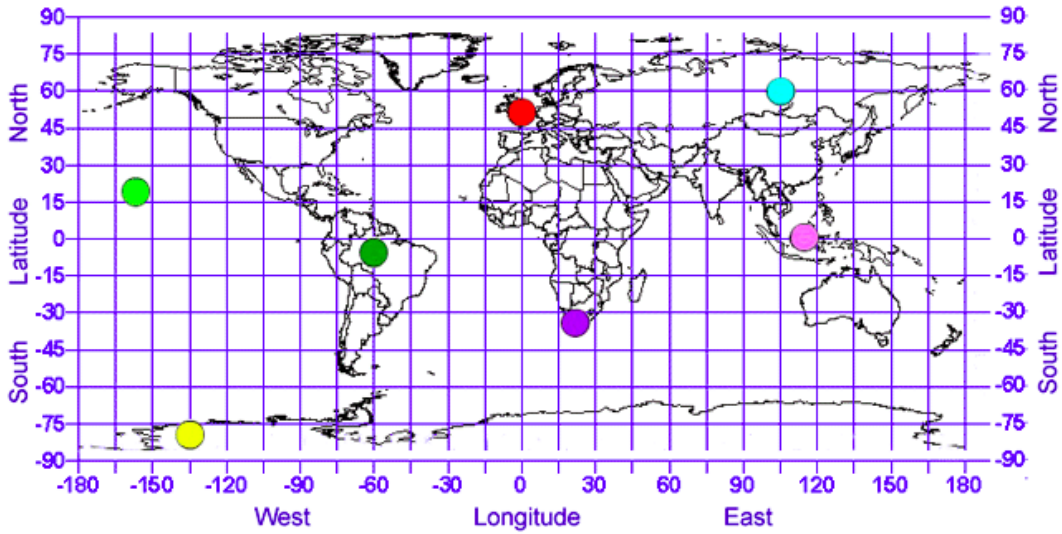
Ordered Pairs: A global address system

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 = ()
type(empty_tuple)

single_number = (3,)
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

Tuples and Numbers?

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

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

second_var = (3,4)
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)
type(empty_tuple)   # <class 'tuple'>
```

A non-empty tuple

```
nonEmpty_tuple = ("a","b","c","d")
nonEmpty_tuple[0]   #   'a'
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')
"Hi" in nonEmpty_tuple   #  True
4 in nonEmpty_tuple   # True
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)

"my" in nonEmpty_tuple   # False
"My" in nonEmpty_tuple   # False
"Hi" in nonEmpty_tuple   # True
"HI" in nonEmpty_tuple   # False

# check to see if detail is in a substring in tuple
"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)
type(s_tuple) # <class 'tuple'>
s_tuple = (1,2,3) + (3,4,5)
s_tuple # (1, 2, 3, 3, 4, 5)
```

tuple to list, remove element, list to tuple

```
s_tuple = (1,2,3)
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)
pair[0] # 3
pair[1] # 4
```

Unpack the contents of a tuple

```
x, y = pair
(x, y) = pair
```

Unpack and perform simultaneous assignment

```
x, y = y, x
(x, y) = (y, x)
```

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)
type(myDictionary_dict)   # <class 'dict'>
```

Dictionaries

Adding to a dictionary

```
myDictionary_dict = {}
myDictionary_dict[0] = "zero"
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']
random.choice(abc_list)    # 'c'
random.choice(abc_list)    # 'd'
```

Choosing Elements from a List

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

Randomly Choosing Elements

Choosing Elements from a Dictionary

```
import random
abc_dict = {1:"one",2:"two",3:"Three"} # {vals : keys}
num_list = list(abc_dict) # convert dict to list
n = random.choice(num_list) # pick a number in list
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()
```

File: genNamesPhoneEmail_i.py

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}")
```

File: genNamesPhoneEmail_i.py

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}")
```

File: genNamesPhoneEmail_i.py