ReDI School of Digital Integration

Welcome

Python Intermediate - Spring 2025

Get to know

Class Overview

- Recap
- Collections
- Lists
- Tuples
- Dictionaries
- Exercises

Recap

ReDI School of Digital Integration

Recap

- **Variable:** Containers for storing data values
  - Variables are created by assigning a value to them: `some_variable = "John Doe"`
- **Built-In Data Types:** Variables have always a type (which can change)
  - String: `a = "Some text"`
  - Integer: `b = 3`
  - Float: `c = 3.14159`
  - Boolean: `d = True`
- **Conditions:** Commands for handling decisions

```
if 4 > 5:
    print("A")
elif True:
    print("B")
else:
    print("C")
```

- **Loops:** To iterate over a sequence

```
for i in range(3):
    print(i)
```

Programming Theory

- Algorithms:
    - A set of rules or steps used to solve a problem
    - Examples: A cooking recipe or an average calculator
- Data Structures:
    - A particular way of organizing data in a computer
    - Examples: Lists, tuples, dictionaries and sets

ReDI School of Digital Integration

Collections

- **Motivation:** Often a “normal” variable is not enough to represent your data
  - Example: How to represent a shopping list for a supermarket?
  - String:

```
shopping_list = "Cheese Bread Milk Eggs Butter"
```

  - How to manipulate? Delete Bread?
- **Definition:** A collection is a grouping of some variable number of data items
- **Advantages:** A collection is useful because we can carry many values around in one convenient package
  - Example: List with items
  - List:

```
shopping_list = ["Cheese", "Bread", "Milk", "Eggs", "Butter"]
```

  - Delete item:

```
shopping_list.remove("Bread")
```

Lists

ReDI School of Digital Integration

Lists

- Lists are changeable (mutable) sequences, typically used to store collections of similar (homogenous) items
- Lists as many other data structures can be created in many ways. In the following the two most common:
  - With the list() function you can create an empty list:

    ```
    shopping_list = list()
    ```

  - With square brackets:

    ```
    shopping_list = ["Cheese", "Bread", "Milk", "Eggs", "Butter"]
    ```

- Items in a list can be accessed with square brackets and passing the index number:
  - Get third item on the list:

    ```
    shopping_list[2]
    ```

ReDI School of Digital Integration

Lists: Properties

- Ordered

```
abc = ["a", "b", "c"]
                        }   abc != cba
cba = ["c", "b", "a"]
```

- Can contain arbitrary objects

```
random_list = ["Milk", 3, 2.3, shopping_list]
```

- Mutable (changeable) and dynamic (changes in size, memory usage)

```
random_list.append("Bread")

random_list[2] = "Eggs"

del random_list[3]
```

ReDI School of Digital Integration

Lists: Slicing

- Slicing is used to get elements of a list in a range
- The format for list slicing is [start:stop:step]
- The sliced list will then go from start to stop-1
- Examples:

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(numbers[4])

print(numbers[4:])

print(numbers[4:7])

print(numbers[1:9:2])

print(numbers[::])
```

ReDI School of Digital Integration

Lists: Useful methods

- **append(**element**)** → Adds an element at the end of the list
- **count(**element**)** → Returns the number of elements with the specified value
- **index(**element**)** → Returns the index of the first element with the specified value
- **insert(**element**)** → Adds an element at the specified position
- **remove(**element**)** → Removes the first item with the specified value
- **reverse()** → Reverses the order of the list
- **sort()** → Sorts the list

ReDI School of
Digital Integration

Lists (more methods)

- Concatenation

```
abc = ['a', 'b', 'c']

defg = ['d', 'e', 'f', ‘g’]
```

$\}$ abc + defg -> ['a', 'b', 'c', 'd', 'e', 'f', 'g']

- Length

```
len(abc)
```

- Math

```
sum([1, 3, 5])

min([1, 3, 5])

max([1, 3, 5])
```

Tuples

ReDI School of Digital Integration

Tuples

- Tuples are not changeable (immutable) sequences, typically used to store collections of heterogeneous data
- Tuples as many other data structures can be created in many ways. In the following the two most common:
  - With round brackets:

    ```
    person = ("Female", 37, "Anna")
    ```

  - With tuple() function:

    ```
    person = tuple("abc")
    ```

- Items in a tuple can be accessed with square brackets and passing the index number:

```
print("Gender: " + person[0] + "\nAge: " + str(person[1]) + "\nName: " +
person[2])
```

Tuples: Properties

- Ordered
- Can contain arbitrary objects
- Immutable (Not changeable)

```
person[2] = "Ana"
```

 TypeError: 'tuple' object does not support item assignment

- Not dynamic

Tuples: Slicing

- Same as with lists

```
print(letters[0:2])
```

ReDI School of Digital Integration

Tuple: Methods

- count(): Returns the number of times a specified value occurs in a tuple
- index(): Searches the tuple for a specified value and returns the position of where it was found

- Concatenation

```
abc = ('a', 'b', 'c’)

defg = ('d', 'e', 'f', ‘g’)
```

- Length

```
len(abc)
```

- Math

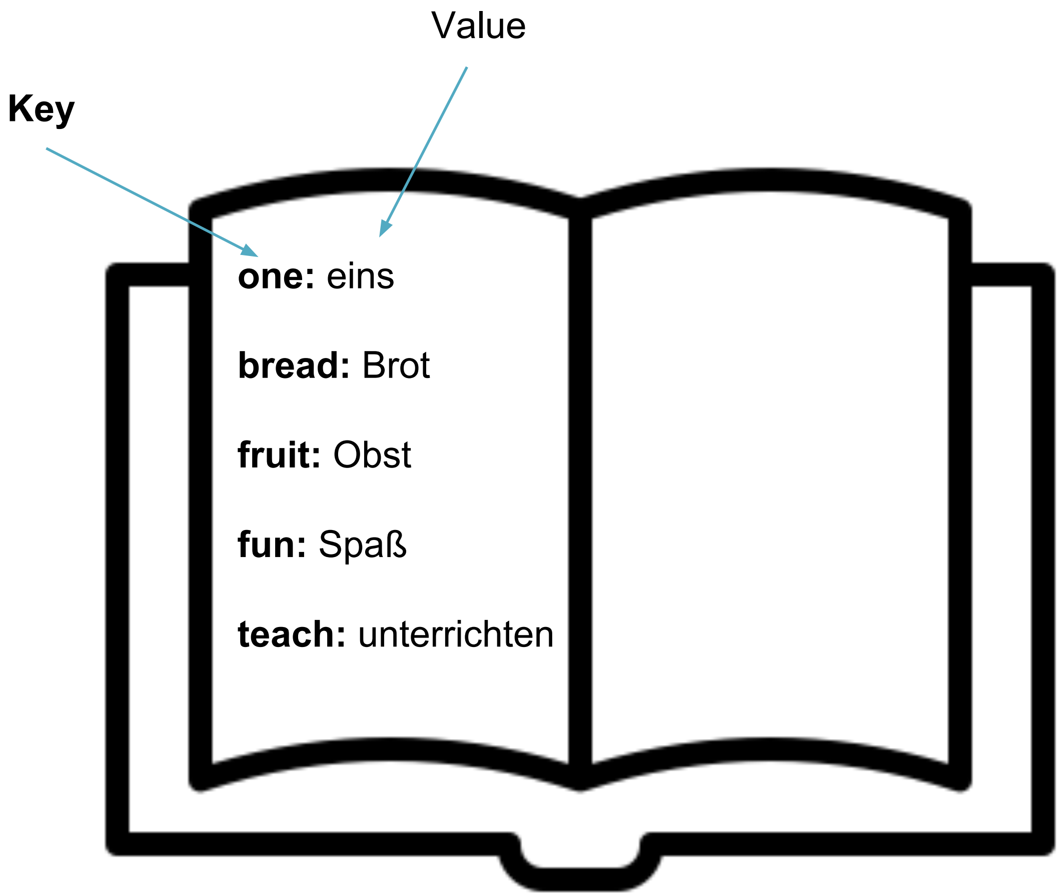
```
sum((1, 3, 5))

min((1, 3, 5))

max((1, 3, 5))
```

Dictionaries

ReDI School of Digital Integration

Intuition

- **What is a dictionary?**



one:	eins
bread:	Brot
fruit:	Obst
fun:	Spaß
teach:	unterrichten

Dictionaries

- A dictionary is a collection of **key-value pairs** which is ordered, changeable and do not allow duplicates.

- Dictionaries can be created with:
  - Curly brakets, keys and values:

    ```
    animal = {"Type": "Mammal", "Legs": 4, "Name": "Elephant"}
    ```

  - With the dict() constructor:

    ```
    animal = dict(Type="Mammal", Legs=4, Name="Elephant")
    ```

- Values can be accessed by passing the key:

  ```
  animal["Name"]
  ```

ReDI School of Digital Integration

Dictionaries Properties

- Ordered

```
d1 = {"a": 1, "b": 2, "c": 3}
```

- Changeable

```
d1["a"] = 0
```

- Dynamic in size

```
d1["d"] = 4

d1.update({"e": 5})
```

- Duplicates Not Allowed (Values get re-written)

```
d1["d"] = 99
```

- Can contain arbitrary objects (even data structures as values)

```
d2 = {
    1: "a",
    "b": 2,
    2.4: 0x4567,
    "the abc": ["a", "b", "c", "d"],
    "a dict": {
        1: "a",
        2: "b"
    }
}
```

Useful Methods

- **clear()** → Removes all key-value pairs from the dictionary.
- **get(key, default=None)** → Returns the value for a key, or a default if the key is not found.
- **keys()** → Returns a list-like view of all keys.
- **values()** → Returns a list-like view of all values.
- **items()** → Returns a list-like view of key-value pairs as tuples.
- **pop(key, default=None)** → Removes and returns the value of a key, or returns default if the key is missing.
- **update(dict2)** → Adds or updates key-value pairs from another dictionary.

Iterables

ReDI School of Digital Integration

Ranges

- The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops.

- Ranges can be initialized with the function range(start, stop[, step])

```
odd_numbers = range(1, 11, 2)
```

Iterables

- **Definition:** An iterable is any Python object capable of returning its members one at a time, permitting it to be iterated over in a for-loop.
- List, tuples, dictionaries and sets are iterables!
- Example 1: Print odd numbers from 1 to 9:

```
for i in range(1, 11, 2):
    print(i)
```

- Example 2: Print names of list:

```
names_list = ["Noah", "Emma", "Oliver", "Charlotte", "Elijah", "Amelia",
"James" ,"Ava"]

for names in names_list:
    print(names)
```

Mnemonic **variables**: serve as placeholders for each item in the iteration.

ReDI School of
Digital Integration

Dictionaries

- Iterate over keys:

```
for key in elephant:
    print(key)
```

- Iterate over values:

```
for value in elephant.values():
    print(value)
```

- Iterate over keys and values

```
for key, value in elephant.items():
    print(key, ": ", value)
```

Exercises

ReDI School of Digital Integration

Exercise 1 (lists):

Create a shopping list with 5 elements:

- Define an empty shopping list using the function command
- Add two elements “Cake” and “Napkins” to the shopping list, by using `insert(position, "Element")`
- Add two more elements [“Balloons”, “Confetti”] using `append("Element")`
- Sort the shopping list alphabetically
- Remove the second element from your list

ReDI School of Digital Integration

Exercise 2

1. How many times does each word appear in the following scientific text? Create a list of tuples with the following structure: [("word_1", count_1), ("word_2", count_2), ...]

> We study methods for estimating model uncertainty for neural networks (NNs) in regression. To isolate the effect of model uncertainty, we focus on a noiseless setting with scarce training data. We introduce five important desiderata regarding model uncertainty that any method should satisfy. However, we find that established benchmarks often fail to reliably capture some of these desiderata, even those that are required by Bayesian theory. To address this, we introduce a new approach for capturing model uncertainty for NNs, which we call Neural Optimization-based Model Uncertainty (NOMU). The main idea of NOMU is to design a network architecture consisting of two connected sub-NNs, one for model prediction and one for model uncertainty, and to train it using a carefully-designed loss function. Importantly, our design enforces that NOMU satisfies our five desiderata. Due to its modular architecture, NOMU can provide model uncertainty for any given (previously trained) NN if given access to its training data. We evaluate NOMU in various regressions tasks and noiseless Bayesian optimization (BO) with costly evaluations. In regression, NOMU performs at least as well as state-of-the-art methods. In BO, NOMU even outperforms all considered benchmarks.

   - Step 1: Convert text to a list of words with .split()
   - Step 2: Create a list with all unique words
   - Step 3: Iterate over all unique words, count occurrences and save into a list

2. Print all the words that appear more than three times in the text
3. Do 1. and 2. using a dictionary, i.e. create a dict like {"word1": count_1, "word2": count_2, ...}

Nested Data Structures

- Nested data structures are **data structures inside other data structures**. They allow complex, hierarchical data representation.
- Examples:
    - Nested dictionary:

```
zoo = {
  "elephant": {
    "species": "African Elephant",
    "weight_kg": 6000
  },
  "tiger": {
    "species": "Bengal Tiger",
    "weight_kg": 220
  }
}
```

    - List of dict:

```
students = [
  {"name": "Alice", "grades": [85, 90, 78]},
  {"name": "Bob", "grades": [92, 88, 84]}
]
```

ReDI School of
Digital Integration

Exercise 3

We have a list of students and their grades:

```
[{'name': 'Student_1', 'grades': [55, 87, 59, 92, 77]},

{'name': 'Student_2', 'grades': [69, 62, 87, 67, 98],

…}
```

1. Calculate the average for each student and add it to the dictionary
2. Calculate the average for all students

Before you go…

Please take 2 minutes to give us feedback on today's lesson.

This feedback is **anonymous** and only seen by **the DCP team**!

Scan the QR code or go to:

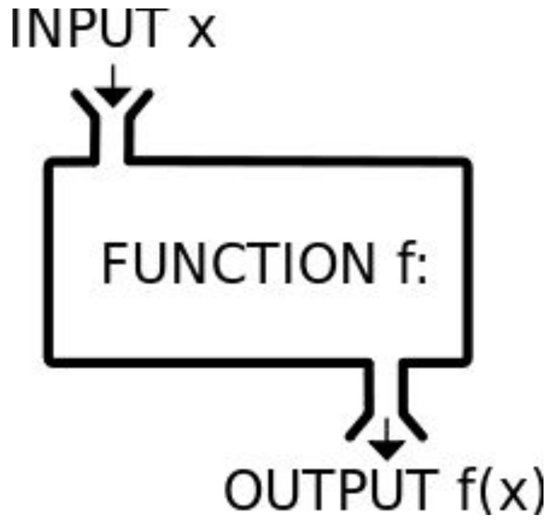
Questions?

ReDI School of Digital Integration

Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- You already know many functions like `print(…)`
- In Python a function is defined using the `def` keyword
- Example 1: Function with return value:

```
def sum_three_numbers(x, y, z):
    return x+y+z


result = sum_three_numbers(1, 2, 3)
print(result)
```



Functions (continued)

- Example 2: Function without return value

```
def print_happy_hollidays_email(sender, recipient):
    print("Dear Mr. " + recipient +
      "\n I wish you happy hollidays" +
      "\n Best regards," +
      sender)


print_happy_hollidays_email("Jim", "Michael")
```