Python

PCEP-PREP
DAY 3

Estevan A.

Subjects	Goals
Data Collections - Tuples, Dictionaries, Lists, Strings	• Construct Vectors, indexing and slicing, basic list methods • Iteraint through list with the for loop, initialize loops in and not in operators list comprehensions • Colles and process data using tuples • Collect and process data using dictionaries • Operate with strings
Functions and Exceptions	• Decompose the code using functions • Oraganize interaction between the function and its enviroment • Python Built in Exceptions Hierarchy • Basics of Pyton Expection Handling •
Exercises/Review	• complete exercises given in class

6 STANDARD DATA TYPES

- Numbers
- String
- List
- Tuple
- Dictionary
- Boolean

List

A list contains items separate by commas and enclosed within square brackets [].

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

OUTPUT

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Exercise 1

Module 1

1. Declare a list with 5 named **colors**
2. Use a loop to iterate through the list to print all elements into console
3. Add 2 more colors into the list
4. Change the 1st index with last index elements

Tuples

A tuple is another sequence data type like a list but is enclosed with ( ) instead. Key difference between List vs Tuples is that list are able to change size and elements where tuples cannot. It can be seen as read-only.

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print tuple               # Prints the complete tuple
print tuple[0]            # Prints first element of the tuple
print tuple[1:3]          # Prints elements of the tuple starting from 2nd till 3rd
print tuple[2:]           # Prints elements of the tuple starting from 3rd element
print tinytuple * 2       # Prints the contents of the tuple twice
print tuple + tinytuple   # Prints concatenated tuples
```

OUTPUT

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

Exercise 2

Module 1

1. create a tuple of letters and print one item
2. unpack a tuple into several variables
3. add an item to a tuple
4. convert a tuple to a string

String

Strings in Python are identified as a contiguous set of characters represented in quotation marks

```
str = 'Hello World!'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

OUTPUT

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

word = "google.com"

Exercise 3

Module 1

1. calculate the length of a string and save the length to variable **word_size**
2. program to count the number of characters (character frequency) in a string

Dictionary

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

Dictionary are kind of hast table type where it consist of key-value pairs. It can be almost any python type but usually numbers or strings. Dictionaries are enclosed with `{}` and values can be assigned or accessed with `[]`.

OUTPUT

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

Exercise 4

Module 1

dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}

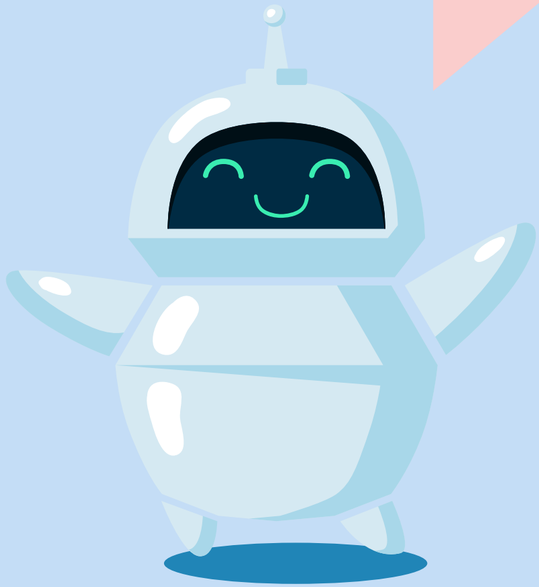
1. Write a script to concatenate the following dictionaries to create a new one
2. Write a script to check whether a given key already exists in a dictionary

Functions

A function is a named sperate part of the code that can be activated on demand.
It can do three of the following:

- Perform an action
- Return a result
- or both

Write a function that returns true if the two given integer values are equal or their sum or difference is 5



Functions cont.

- Activating a function is done by the **function invocation**
- A function can be equipped with an arbitrary number of **parameters.**
- If function is supposed to evaluate a result it must prefor the **return** expression.
-

```
def function(parameter):
    if parameter == False:
        return True


print(function(False), function(True))
```

Functions cont.

- A function can declare **default values** for some or all of its parameters
- The **positional** parameter passing technique.
- The **keyword** parameter passing technique is a technique based on the assumption that the arguments are associated with the parameters based upon names.

```
def function(parameter = False):
    return parameter


print(function(True), function())
```

```
def function(a, b, c):
    print(a, b, c)


function(1, 2, 3)
```

```
def function(a, b, c):
    print(a, b, c)


function(c=3, a=1, b=2)
```

Exceptions and Debugging

- An **exception** is an event caused by an execution error which can induce program termination if not handled properly.
- To **control** exceptions and to handle them.

```
try:
    x = 1 / 0
except:
    x = None
print('PROCEEDING')
```

Exceptions and Debugging PART 2

Here is a list of the most common Python exceptions:

- `ZeroDivisionError` : raised by a division in which the divider is **zero** or is indistinguishable from zero (/, //, and %)
- `ValueError` : raised by the use of values that are **inappropriate** in the current context, for example, when a function receives an argument of a proper type, but its value is **unacceptable**, for example, int('')
- `TypeError` : raised by attempts to apply data of a **type** which cannot be accepted in the current context, for example, `int(None)`
- `AttributeError` : raised – among other occasions – when the code tries to activate a **method** that doesn't exist in a certain item, for example, `the_list.apend()` (note the typo!)
- `SyntaxError` : raised when the control reaches a line of code that **violates** Python's grammar, and which has remained undetected until now;
- `NameError` : raised when the code attempts to make use of a **non-existent** (not previously defined) **item**, for example, a variable or a function.

Exceptions and Debugging PART 3

When more than one exception is expected inside the try block and these different exceptions require different handling, another syntax is used where there is more than one named except branch.

```
try:
    print(1 / int(input("Enter a number: ")))
except ValueError:
    print('NAN')
except ZeroDivisionError:
    print('ZERO')
except:
    print('ERR')
```

Exceptions and Debugging PART 4

An error existing in the code is commonly called a **bug**.

The process by which bugs are detected and removed from the code is called **debugging**.

The tool which allows the programmer to run the code in a fully controllable environment is called a **debugger**.

The '**print debugging**' technique is a trivial debugging technique in which the programmer adds some `print()` function invocations which help to trace execution paths and output the values of selected critical variables.

The process in which the code is probed to ensure that it will behave correctly in a production environment is called **testing**. The testing should prove that all execution paths have been executed and caused no errors.

The programming technique in which the tests and test data are created before the code is written or created in parallel with the code is called **unit testing**. It is assumed that every code amendment (even the most trivial) is followed by the execution of all previously defined tests.

Exercise 5

Module 1

1. Write a Python function to find the maximum of three numbers.
2. Write a Python function to sum all the numbers in a list.
3. Write a Python function to check whether a number falls within a given range.