Python (CIE 1)

Tuples

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is **ordered** and **unchangeable**. (immutable)
- Tuples are written with round brackets.
  Ex: thistuple = ("apple", "banana", "cherry")
- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
- Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Create Tuple With One Item
To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognise it as a tuple.
From Python's perspective, tuples are defined as objects with the data type 'tuple':

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

You can access tuple items by referring to the index number, inside square brackets:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Negative indexing means start from the end.
-1 refers to the last item, -2 refers to the second last item etc.

You can specify a range of indexes by specifying where to start and where to end the range.
When specifying a range, the return value will be a new tuple with the specified items.
Note: The search will start at index 2 (included) and end at index 5 (not included)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

By leaving out the start value, the range will start at the first item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

By leaving out the end value, the range will go on to the end of the list:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

Specify negative indexes if you want to start the search from the end of the tuple:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

Check if Item Exists
```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example

Convert the tuple into a list to be able to change it:
```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.
1. **Convert into a list**: Just like the workaround for *changing* a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

Example

Convert the tuple into a list, add "orange", and convert it back into a tuple:
```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

2. **Add tuple to a tuple**. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

Example

Create a new tuple with the value "orange", and add that tuple:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

Example

Packing a tuple:

```
fruits = ("apple", "banana", "cherry")
```

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

Example

Unpacking a tuple:

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

Note: The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

Using Asterisk*

Only 1 * could be used

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

Example

Assign the rest of the values as a list called "red":

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits
print(green)
print(yellow)
print(red)
```

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

Example

Add a list of values the "tropic" variable:

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
(green, *tropic, red) = fruits
print(green)
print(tropic)
```

```
print(red)
```

Loop Through a Tuple

You can loop through the tuple items by using a for loop.

Example

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Loop Through the Index Numbers

You can also loop through the tuple items by referring to their index number.
Use the range() and len() functions to create a suitable iterable.

Example

Print all items by referring to their index number:

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

Using a While Loop

You can loop through the list items by using a while loop.
Use the len() function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by refering to their indexes.
Remember to increase the index by 1 after each iteration.

Example

Print all items, using a while loop to go through all the index numbers:

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

A programming language is object-oriented if it focuses design around data and objects, rather than functions and logic. On the contrary, a programming

language is procedure-oriented if it focuses more on functions (code that can be reused).

Python Features and Advantages

14 Most Important Python Features and How to Use them?

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement. if statement, for statement, while statement, etc. are other kinds of statements.

Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data that can be changed later in the program. For example,
number = 10

Python Modules

Python Modules

Encapsulation

It acts as a protective shield that puts ***restrictions on accessing variables and methods directly***, and can prevent accidental or unauthorised modification of data.

Access modifiers

Access modifiers limit access to the variables and functions of a class. Python uses three types of access modifiers; they are -

***private*, *public* and *protected*.**

https://www.educative.io/edpresso/what-is-encapsulation-in-python

From notes

1. Classes
2. Method Overloading and Overriding
3. List
4. Loops in Python
5. Dictionary
6. Function
7. Recursion
8. Lambda: https://www.youtube.com/watch?v=dX6lWSp7pP4
9. Map: https://www.geeksforgeeks.org/python-map-function/
10. Filter: https://www.youtube.com/watch?v=kYIrDMbqunw
11. Exception Handling: https://www.programiz.com/python-programming/exception-handling

Arguments of Function

https://levelup.gitconnected.com/5-types-of-arguments-in-python-function-definition-e0e2a2cafd29

Class Attributes vs Instance Attributes:

https://www.tutorialsteacher.com/articles/class-attributes-vs-instance-attributes-in-python

Python Sets

https://www.programiz.com/python-programming/set

Difference between Structured Programming and Object Oriented Programming

Difference between Structured Programming and Object Oriented Programming - GeeksforGeeks

Python (CIE 2)

Class , Class Attribute | Instance , Instance Attribute

https://dzone.com/articles/python-class-attributes-vs-instance-attributes

Encapsulation in Python

https://www.geeksforgeeks.org/encapsulation-in-python/

Polymorphism

https://www.programiz.com/python-programming/polymorphism

Operator Overloading

https://www.programiz.com/python-programming/operator-overloading

Function Overloading

Notes

Constructors

https://www.geeksforgeeks.org/constructors-in-python/

Destructors

https://www.geeksforgeeks.org/destructors-in-python/

Modules

- https://www.geeksforgeeks.org/python-modules/
- Notes

Packages

https://www.geeksforgeeks.org/python-packages/

Inheritance

Notes

Error and Exception

- https://www.programiz.com/python-programming/exceptions
- https://www.w3schools.com/python/python_try_except.asp

S.no.	On the basis of	Procedural Programming	Object-oriented programming
1	**Definition**	It is a programming language that is derived from structure programming and based upon the concept of calling procedures. It follows a step-by-step approach in order to break down a task into a set of variables and routines via a sequence of	Object-oriented programming is a computer programming design philosophy or methodology that organizes/ models software design around data or objects rather than functions and logic.
2	**Security**	It is less secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
3	**Approach**	It follows a top-down approach.	It follows a bottom-up

4	**Data movement**	In procedural programming, data moves freely within the system from one function to another.	In OOP, objects can move and communicate with each other via member functions.
5	**Orientation**	It is structure/ procedure-oriented.	It is object-oriented.
6	**Access modifiers**	There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
7	**Inheritance**	Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
8	**Code reusability**	There is no code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.
9	**Overloading**	Overloading is not possible in procedural programming.	In OOP, there is a concept of function overloading and operator
10	**Importance**	It gives importance to functions over	It gives importance to data over
11	**Virtual class**	In procedural programming, there are no virtual classes.	In OOP, there is an appearance of virtual classes in inheritance.
12	**Complex problems**	It is not appropriate for complex	It is appropriate for complex problems.
13	**Data hiding**	There is not any proper way for data hiding.	There is a possibility of data hiding.

14	**Program division**	In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
15	**Examples**	Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++

Python (CIE 3)

File Operations

Python File I/O: Read and Write Files in Python

Opening Files in Python
Python has a built-in open() function to open a file. This function returns a file object, also called a **handle**, as it is used to read or modify the file.
>>> f = open("test.txt")
We specify whether we want to read **r**, write **w** or append **a** to the file.
The default is reading in text mode.

Ex:
f = open("test.txt") # equivalent to 'r' or 'rt'
f = open("test.txt",'w') # write in text mode
f = open("img.bmp",'r+b') # read and write in binary mode

Closing Files in Python
Closing a file will free up the resources that were tied with the file. It is done using the close() method available in Python.
Python has a garbage collector to clean up unreferenced objects but we must not rely on it to close the file.

EX:

```
f = open("test.txt", encoding = 'utf-8')
# perform file operations
f.close()
```

Writing to Files in Python

In order to write into a file in Python, we need to open it in write w, append a or exclusive creation x mode.

Writing a string using the write() method. This method returns the number of characters written to the file.

EX:

```
with open("test.txt",'w',encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
```

This program will create a new file named test.txt in the current directory if it does not exist. If it does exist, it is overwritten.

Reading Files in Python

To read a file in Python, we must open the file in reading r mode.

We can use the read(size) method to read in the size number of data.

If the size parameter is not specified, it reads and returns up to the end of the file.

We can read the **text.txt** file in following way

EX:

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4)    # read the first 4 data
'This'

>>> f.read(4)    # read the next 4 data
' is '

>>> f.read()     # read in the rest till end of file
'my first file\nThis file\ncontains three lines\n'

>>> f.read()  # further reading returns empty sting
''
```

We can see that the read() method returns a newline as '\n'. Once the end of the file is reached, we get an empty string on further reading.

Magic methods

Magic or Dunder Methods in Python

Python Decorators

https://youtu.be/8hWlWyBfdQE?list=PL98qAXLA6afuh50qD2MdAj3ofYjZR_Phn

Python has **decorators** to add functionality to an existing code.
This is also called **metaprogramming** because a part of the program tries to modify another part of the program at compile time.

Python Decorators: How to Use it and Why?

Context Manager

Context Manager in Python - GeeksforGeeks

Python Generators

Python yield, Generators and Generator Expressions

Database

Follow the lab program