Functions in Python

- A function is a portion of code within a larger program that performs a specific task.
- Functions are useful in reusing the code and eliminate code redundancy.
- Functions are also used to organize our code into manageable blocks.

Syntax:

```
def function_name(parameters):
        “ “ “Docstring” ” ”
        Statement(s)
    return [expression]
```

Example:

```
def greet( ):
    print(“Hello Everyone!”)
    print(“Good Morning”)

greet( )
```

Output:

Hello Everyone!
Good Morning

Advanced Features of functions

Default Arguments:

- Function arguments can have default values in python.
- When we call a function without a value for an argument, its default value is used if available.
- Otherwise it gives an error.

Example:

1.
```
def Welcome (greet, name = “world”):
    print(greet, name)

Welcome(“Hello”)
```

Output:

Hello world

2.
```
def Welcome(name="World", greet, age=23):
    print(greet,name,age)

Welcome("Hello")
```

Error:

```
def Welcome(name="World",greet, age=23):
                         ^
```

SyntaxError: parameter without a default follows parameter with a default

Keyword Arguments

- No need to remember the order of arguments while calling the functions by passing keyword arguments.
- Instead, we can use name of the argument to pass a value to it.
- Keyword arguments must be specified at the end.

Example:

1.
```
def marks(first, second, third):
    print("first: %d second: %d and third: %d" %(first, second, third))
marks(34,23,45)
```

Output: first: 34 second: 23 and third: 45

2. marks(34,45,23)

Output: first: 34 second: 45 and third: 23

3. marks(34,second = 23, 45)

Output:

```
marks(34,second = 23, 45)
                      ^
```

SyntaxError: positional argument follows keyword argument

Arbitrary arguments

- We may not always know in advance the number of arguments that will be passed into a function.
- Use an asterisk (*) before an argument name to denote arbitrary number of arguments.

Example:

```
1.      def family (* names):
            print(names)
  family (“Duryodhana”, “Dushasana”)

2.      def person(** attributes):
    print(attributes)
person(name = "John", age = 34,height = 182)
```

Output:

('Duryodhana', 'Dushasana')

Output:

{'name': 'John', 'age': 34, 'height': 182}

Built-in functions

- Python also provides built-in functions some of them are:
  - abs( ) - Returns absolute value of the given number
  - any( ) - Returns value true if any of the items in an iterable are true; else returns False
  - dir( ) - Returns all properties and methods of the specified object, without the values.
  - help( ) - Returns a help page with detailed documentation of a particular object passed as a parameter(if any).

Modules in python

- A python module is a python file with .py extension including statements and definitions. It contains code that you can reuse in several programs.
- For example: A file containing python code, demo.py, is called a module, and its module name would be demo
- Provides flexibility to organize the code in a logical way. Modules contents are accessed with *import* statement.

Types of Python Modules:

Built-in modules – are predefined modules that are part of python standard library

Ex: random, datetime, sys

User-defined modules – The user creates the user defined modules to ease complex tasks in a project. You can define your own functions and classes.

Example: demo.py

```
from Calc import *
    a = 9
    b = 7
    c = add(a,b)
    print(c)
```

Calc.py

```
def add (a, b):
    return a + b
def sub (a, b):
    return a  b
def mul (a, b):
    return a * b
def div (a, b):
    return a / b
```

Output:

```
16
```

Data Manipulation in Python

- Data manipulation in Python can be done using various libraries such as Pandas, NumPy, and others. Here's a brief overview using Pandas, one of the most commonly used libraries for data manipulation.
- ## Pandas Basics:

➢ Importing Pandas:

To start working with Pandas, *import* it:

```
import pandas as pd
```

➢ Reading Data:

Pandas can read data from various sources like CSV, Excel, SQL databases, etc.

```
data = pd.read_csv('file.csv')    # Reading a CSV file
```

➢ Exploring Data:

Once you've loaded data, you can perform basic explorations:

```
print(data.head()) # Display the first few rows

print(data.describe()) # Get summary statistics
```

Selecting Data:

You can select specific columns or rows:

```
column_data = data['Column_Name’] # Selecting columns

filtered_data = data[data['Column_Name'] > 10] # Selecting rows based on conditions
```

Data Cleaning:

Pandas offers functions to clean data by handling missing values, duplicates, etc.

```
# Handling missing values
data.dropna()  # Drops rows with missing values
data.fillna(value)  # Fills missing values with a specified value

# Removing duplicates
data.drop_duplicates()
```

Manipulating Data:

You can perform various operations on data:

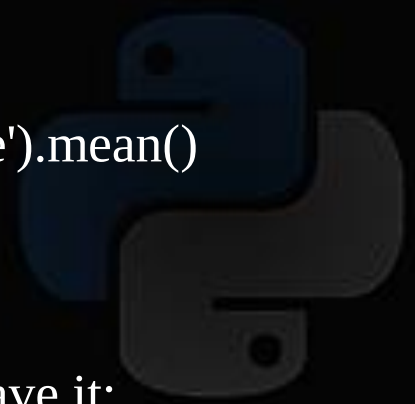
```
# Adding a new column
data['New_Column'] = data['Column1'] + data['Column2']
```

```
print(data.columns) # Check column names

print(data.dtypes) # Check data types

# Applying functions
data['Column'] = data['Column'].apply(my_function)

# Grouping data
grouped_data = data.groupby('Column_Name').mean()
```



Exporting Data:

After manipulating data, you might want to save it:

```
    # Saving to CSV
data.to_csv('new_file.csv', index=False)
```

- Pandas provides numerous functionalities to handle data effectively. This is a basic overview, and there's much more to explore and utilize based on specific data manipulation requirements.

Thank You