

```

git config --global user.name "Your Name"    # Sets Git username
git config --global user.email your@email.com # Sets Git email
git config --list                             # Shows Git settings

```

```

git add <filename>    # Stage one file
git add *              # Stage all files
git commit -m "message" # Commit staged files
git diff              # Show changes

```

```

git branch          # List branches
git branch <name>   # Create branch
git checkout <name> # Switch branch
git merge <branch> # Merge branch

```

```

git remote add origin <URL> # Link repo to GitHub
git remote -v               # Show remote URLs
git remote set-url origin URL # Change GitHub repo

```

```

git push -u origin master # First push
git push                  # Future pushes

```

```

git tag -a v1.0 -m "Initial release" # Create tag
git push origin v1.0                 # Push tag

```

```

git submodule add <URL>    # Add submodule
git clone <URL>             # Clone repo
.gitignore __pycache__/_/

```

```

ascending = sorted(numbers)
or
descending = sorted(numbers, reverse=True)
len(list)    # count elements
sum(list)    # sum all elements
min(list)    # smallest number
max(list)    # largest number
sorted(list) # returns a new sorted list
list.append(item) # add item to end

```

GIT COMMANDS

```

def bubble_sort_ascending(arr):
    for i in range(len(arr)-1):
        for j in range(len(arr)-i-1):
            if arr[j] > arr[j+1]: #put < for descending
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

```

```

def calculate_bmi(height, weight):
    print("Height: " + str(height))
    print("Weight: " + str(weight))

```

```

bmi = weight / (height * height)
print("Your BMI is: " + str(bmi))

```

```

if bmi < 18.5:
    return -1
elif bmi >= 18.5 and bmi <= 25:
    return 0
else:
    return 1

```

```
calculate_bmi(weight=57, height=1.73)
```

```

def calc_average(avg):
    print("calculate average")
    totallen = len(avg)
    totalval = sum(avg)
    avg = totalval / totallen

```

```
return avg
```

```

def find_min_max(minmax):
    minimum = min(minmax)
    maximum = max(minmax)

    return [minimum, maximum]

```

extra

```

def sort_temperature(values):
    newlist = sorted(values)

    return newlist

```

lab2

```

def calc_median_temperature(values):
    vallist = sorted(values)
    total_len = len(vallist)

    if total_len % 2 == 0:
        mid1 = total_len // 2
        mid2 = total_len // 2 + 1
        medium = (vallist[mid1] + vallist[mid2]) / 2
    else:
        middle = total_len // 2
        medium = vallist[middle]

    return medium

```

```

def get_user_input():
    num_list = []
    x = input("Enter a list of number with a ,")
    x = x.split(",")
    for item in x:
        num_list.append(float(item))

    return num_list

```

```

def main():
    print("ET0735 (DevOps for AIoT) -Lab 2-
    Introduction to python")
    display_main_menu()
    num_list = get_user_input()

```

```

if __name__ == "__main__":
    main()

```

How to access values in dictionary
fruit_basket = {'apple': 5, 'banana': 3, 'orange': 7}

#{key, value}
{'apple': 5} **For list**

Access value using key
apple_count = fruit_basket['apple']

Looping through dictionary keys
for fruit in fruit_basket.keys():
 print("-", fruit)

Looping through keys and values together
for fruit in fruit_basket.keys():
 quantity = fruit_basket[fruit]
 print(f"-{fruit}: {quantity}")

price_list={'apple' : 1.20, 'orange':1.40, 'watermelon': 6.50, 'pineapple': 2.70, 'pear' : 0.90, 'papaya': 2.95, 'pomegranate': 4.95 }

quantity_list= {'apple': 5, 'orange':5, 'watermelon': 1, 'pineapple': 2, 'pear' : 10, 'papaya': 1, 'pomegranate': 2}

def total_cost_shipping():
 total_cost = 0 **Code**

for key in price_list.keys():
 if key in quantity_list:
 #getting price for this fruit
 price = price_list[key]
 #getting quantity for this fruit
 quantity = quantity_list[key]

cost = quantity * price

total_cost = total_cost + cost

print(total_cost)
 return total_cost

def cost_of_fruits(fruit, quantity):
 for key in price_list.keys():
 if key == fruit:
 cost = quantity*price_list[key]
 break

 print("cost of ", quantity, fruit, "=", cost)
 return cost

def main():

cost_of_fruits('apple', 10)
 total_cost_shipping()

if __name__ == "__main__":
 main()

import price_info **testing**

def test_total_cost_shipping():
 #arrange
 expected_result = 46.75
 #act
 result = price_info.total_cost_shipping()

#assert
 assert result == expected_result

def test_cost_of_fruits():
 #arrange
 fruit_name = "apple"
 quantity = 10
 expected_result = 12.0

 #act
 result = price_info.cost_of_fruits(fruit_name, quantity)

 #assert
 assert result == expected_result

```

employee_data = [
    {"name": "John", "age": 30, "department": "Sales", "salary": 50000},
    {"name": "Jane", "age": 25, "department": "Marketing", "salary": 60000},
    {"name": "Mary", "age": 23, "department": "Marketing", "salary": 56000},
    {"name": "Chloe", "age": 35, "department": "Engineering", "salary": 70000},
    {"name": "Mike", "age": 32, "department": "Engineering", "salary": 65000},
    {"name": "Peter", "age": 40, "department": "Sales", "salary": 60000}
]

```

```

def get_employees_by_age_range(age_lower_limit, age_upper_limit):
    result = []

```

```

    # check for age limits and append the item to result

```

```

    for item in employee_data:

```

```

        if int(item["age"]) > int(age_lower_limit) and int(item["age"]) < int(age_upper_limit):

```

```

            result.append(item)

```

```

    return result

```

```

def calculate_average_salary():

```

```

    total = 0

```

```

    average = 0

```

```

    for item in employee_data:

```

```

        employee_salary = item["salary"]

```

```

        total = total + employee_salary

```

```

    average = total / len(employee_data)

```

```

    return round(average, 2)

```

```

def get_employees_by_dept(department):

```

```

    result = []

```

```

    for item in employee_data:

```

```

        if item["department"] == department:

```

```

            result.append(item)

```

```

    return result

```

list of dictionaries

```

import employee_info

```

```

def test_get_employeesbyagerange():

```

```

    #arrange

```

```

    employee_data = [

```

```

        {"name": "John", "age": 30, "department": "Sales", "salary": 50000},

```

```

        {"name": "Jane", "age": 25, "department": "Marketing", "salary": 60000},

```

```

        {"name": "Mike", "age": 32, "department": "Engineering", "salary": 65000},

```

```

    ]

```

```

    #act

```

```

    result = employee_info.get_employees_by_age_range(23, 35)

```

```

    #assert

```

```

    assert result == employee_data

```

```

def test_calc_avg_sal():

```

```

    #arrange

```

```

    expected_result = 60166.67

```

```

    #act

```

```

    result = employee_info.calculate_average_salary()

```

```

    #assert

```

```

    assert result == expected_result

```

```

def test_getemployeebydept():

```

```

    #arrange

```

```

    employee_data = [

```

```

        {"name": "Jane", "age": 25, "department": "Marketing", "salary": 60000},

```

```

        {"name": "Mary", "age": 23, "department": "Marketing", "salary": 56000},

```

```

    ]

```

```

    #act

```

```

    result = employee_info.get_employees_by_dept("Marketing")

```

```

    #assert

```

```

    assert result == employee_data

```

```
def get_longest_workout(workouts):
    # If there are no workouts, return None
    longest = None
    for workout in workouts:
        if longest == None:
            longest = workout
        elif workout["duration"] > longest["duration"]:
            longest = workout

    print(longest)
    return longest
```

```
def calc_total_duration(workouts):
    total = 0
    for workout in workouts:
        total_duration = workout["duration"]
        total = total + total_duration
    # Add your implementation from here
    #HINT: start with this code: for workout in workouts:
    print(total)
    return total
```

```
def calc_average_duration(workouts):
    average = 0

    average = calc_total_duration(workouts) / len(workouts)
    # Add your implementation from here
    #HINT: Use calc_total_duration(workouts) and len(workouts)

    print(average)
    return average
```

Prac paper

```
import labsample1
```

```
def test_get_longest_workout():
    workouts = labsample1.load_csv()
    #Arrange
    expected = {'date': '25.01.2022', 'activity': 'Cycling', 'duration': 75.0}
    #Act
    test = labsample1.get_longest_workout(workouts)
    #assert
    assert(test == expected)
```

```
def test_calc_total_duration():
    workouts = labsample1.load_csv()
    #arrange
    expected = 853.0
    #act
    test = labsample1.calc_total_duration(workouts)
    #assert
    assert(test == expected)
```

```
def test_calc_average_duration():
    workouts = labsample1.load_csv()
    #arrange
    expected = 42.65
    #act
    test = labsample1.calc_average_duration(workouts)
    #assert
    assert(test == expected)
```

```
def calc_mdeian_duration(workouts):
    median = 0
    sortmed = []
    for workout in workouts:
        sortmed.append(workout["duration"])
    sortingmed = sorted(sortmed)

    total_len = len(sortingmed)

    if total_len % 2 == 0:
        mid1 = total_len // 2
        mid2 = total_len // 2 + 1
        medium = (sortingmed[mid1] + sortingmed[mid2]) / 2
    else:
        middle = total_len // 2
        medium = sortingmed[middle]
    return medium
```