Data Types

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Variables

- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.
- Variables do not need to be declared with any particular type, and can even change type after they have been set.
- A variable name cannot be any of the Python keywords.
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

In [1]:
```
x = 5
y = "Hello, World!"
```

Casting

In [2]:
```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

In [3]:
```
print("x is: " , type(x))
print("y is: " , type(y))
print("z is: " , type(z))
```

```
x is:  <class 'str'>
y is:  <class 'int'>
z is:  <class 'float'>
```

Case sensitivity

In [4]:
```
a = 4
A = "John"
# A will not overwrite a
```

Assigning multiple variables at once

In [5]:
```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

In [6]:
```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
apple
banana
cherry
```

Strings

In [7]:
```
# Strings are arrays
s = "Hello world!"
print(s[0])
```

```
H
```

In [8]:
```
for x in "banana":
    print(x)
```

```
b
a
n
a
n
a
```

In [9]:
```
txt = "Regression analysis section one"
print("section" in txt)
```

```
True
```

In [10]:
```
print("two" not in txt)
```

```
True
```

In [11]:
```
if "two" not in txt:
    print("The word 'two' is not present")
```

```
The word 'two' is not present
```

Slicing

In [12]:
```
b = "Hello world!"
print(b[:4])
```

```
Hell
```

Modifying strings

In [13]:
```
a = " Hello world! "
print(a.upper())
print(a.lower())
print(a.strip())
print(a.replace("H", "J"))
print(a.split(" "))
```

```
 HELLO WORLD!
 hello world!
Hello world!
 Jello world!
['', 'Hello', 'world!', '']
```

F-String

In [14]:
```
price = 59
txt = f"The price is {price} dollars"
print(txt)
```

```
The price is 59 dollars
```

Operators

Arithmetic Operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

Comparison Operators

Operator	Name
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3

In [15]:
```
x = 5 # 101
x = x | 3 # 101 or 011 = 111 = 7
x
```

Out[15]: 7

In [16]:
```
x = 5 # 101
x = x & 3 # 101 and 011 = 001 = 1
x
```

Out[16]: 1

In [17]:
```
x = 5 # 101
x = x ^ 3 # 101 XOR 011 = 110 = 6
x
```

Out[17]: 6

Lists

- List items are ordered, changeable, and allow duplicate values.
- A list can contain different data types.

In [18]:
```
list1 = ["abc", 34, True, 40, "male", "abc"]
print(list1)
```

```
['abc', 34, True, 40, 'male', 'abc']
```

In [19]:
```
list2 = list(("abc", 34, True, 40, "male", "abc"))
print(list2)
```

```
['abc', 34, True, 40, 'male', 'abc']
```

In [20]:
```
fruits = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
fruits[1:3] = ["orange", "watermelon"]
print(fruits)
```

```
['apple', 'orange', 'watermelon', 'orange', 'kiwi', 'mango']
```

In [21]:
```
fruits.append("strawberry")
print(fruits)
```

```
['apple', 'orange', 'watermelon', 'orange', 'kiwi', 'mango', 'strawberry']
```

In [22]:
```
fruits.insert(3, "blueberry")
print(fruits)
```

```
['apple', 'orange', 'watermelon', 'blueberry', 'orange', 'kiwi', 'mango',
'strawberry']
```

In [23]:
```
summer = ["mango", "watermelon"]
fruits = ["orange", "strawberry", "grapes"]
fruits.extend(summer)
fruits
```

Out[23]:
```
['orange', 'strawberry', 'grapes', 'mango', 'watermelon']
```

In [24]:
```
fruits.remove("grapes")
fruits
```

Out[24]:
```
['orange', 'strawberry', 'mango', 'watermelon']
```

In [25]:
```
fruits.pop(2)
fruits
```

Out[25]:
```
['orange', 'strawberry', 'watermelon']
```

In [26]:
```
del fruits[0]
fruits
```

Out[26]:
```
['strawberry', 'watermelon']
```

List comprehension

newlist = [expression for item in iterable if condition == True]

In [27]:
```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [28]:
```
newlist = [x for x in fruits if "a" in x]
newlist
```

Out[28]:
```
['apple', 'banana', 'mango']
```

List methods

In [29]:
```
list1 = [34, 2, 42, 10]
list1.sort()
print(list1)
```

```
[2, 10, 34, 42]
```

In [30]:
```
list1 = [34, 2, 42, 10]
list1.sort(reverse=True)
print(list1)
```

```
[42, 34, 10, 2]
```

In [31]:
```
list1 = [34, 2, 42, 10]
list2 = list1.copy()
list2.sort()
print(list1)
print(list2)
```

```
[34, 2, 42, 10]
[2, 10, 34, 42]
```

In [32]:
```
list1 = [34, 2, 42, 10]
list1.clear()
print(list1)
```

```
[]
```

In [33]:
```
list1 = [34, 2, 42, 10]
list1.reverse()
print(list1)
```

```
[10, 42, 2, 34]
```

Tuples

- Tuple items are ordered, unchangeable, and allow duplicate values.
- Accessing an item is just as a list.

In [34]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

In [35]:
```
# Tuple items are immutable but we can workaround to change it
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
('apple', 'kiwi', 'cherry')
```

In [36]:
```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

```
('apple', 'banana', 'cherry', 'orange')
```

Sets

- Set items are unordered, unchangeable, and do not allow duplicate values.
- You can add/remove items but cannot change any

In [37]:
```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

```
{'cherry', 'apple', 'banana'}
```

In [38]:
```
thisset = {"apple", "banana", "cherry", True, 1, 2} # True and 1 will be co
nsidered the same thing
print(thisset)
```

```
{True, 2, 'banana', 'apple', 'cherry'}
```

In [39]:
```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
    print(x)
```

```
cherry
apple
banana
```

In [40]:
```
thisset.add("orange")
print(thisset)
```

```
{'cherry', 'apple', 'orange', 'banana'}
```

In [41]:
```
thisset.remove("cherry")
print(thisset)
```

```
{'apple', 'orange', 'banana'}
```

In [42]:
```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2) # or set3 = set1 | set2
print(set3)
```

```
{'a', 1, 'c', 2, 3, 'b'}
```

In [43]:
```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
{'a', 1, 'c', 2, 3, 'b'}
```

In [44]:
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.intersection(set2) # or set3 = set1 & set2
print(set3)
```

```
{'apple'}
```

In [45]:
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.difference(set2)

print(set3)
```

```
{'cherry', 'banana'}
```

In [46]:
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set2.difference(set1)

print(set3)
```

```
{'google', 'microsoft'}
```

Dictionaries

- A dictionary is a collection which is ordered, changeable and do not allow duplicates.

In [47]:
```
thisdict = {
  "name": "John",
  "gender": "Male",
  "age": 25
}
print(thisdict)
```

```
{'name': 'John', 'gender': 'Male', 'age': 25}
```

In [48]:
```
thisdict["name"]
```

Out[48]:
```
'John'
```

In [49]:
```
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

```
{'name': 'John', 'age': 36, 'country': 'Norway'}
```

In [50]:
```
x = thisdict.get("age")
print(x)
```

```
36
```

In [51]:
```
x = thisdict.keys()
print(x)
```

```
dict_keys(['name', 'age', 'country'])
```

In [52]:
```
x = thisdict.values()
print(x)
```

```
dict_values(['John', 36, 'Norway'])
```

In [53]:
```
thisdict["country"] = "Egypt"
print(thisdict)
```

```
{'name': 'John', 'age': 36, 'country': 'Egypt'}
```

In [54]:
```
thisdict["job"] = "Engineer"
print(thisdict)
```

```
{'name': 'John', 'age': 36, 'country': 'Egypt', 'job': 'Engineer'}
```

In [55]:
```
thisdict.pop("country")
print(thisdict)
```

```
{'name': 'John', 'age': 36, 'job': 'Engineer'}
```

In [56]:
```
myfamily = {
  "child1" : {
    "name" : "Ahmed",
    "year" : 2004
  },
  "child2" : {
    "name" : "Mohamed",
    "year" : 2007
  },
  "child3" : {
    "name" : "Mahmoud",
    "year" : 2011
  }
}
myfamily
```

Out[56]:
```
{'child1': {'name': 'Ahmed', 'year': 2004},
 'child2': {'name': 'Mohamed', 'year': 2007},
 'child3': {'name': 'Mahmoud', 'year': 2011}}
```

Some methods

In [57]:
```
thisdict = dict(name = "John", age = 36, country = "Norway")
thisdict.clear()
print(thisdict)
```

```
{}
```

In [58]:
```
dict1 = dict(name = "John", age = 36, country = "Norway")
dict2 = dict1.copy()
dict1["name"] = "Mark"
print(dict1)
print(dict2)
```

```
{'name': 'Mark', 'age': 36, 'country': 'Norway'}
{'name': 'John', 'age': 36, 'country': 'Norway'}
```

If .. else

In [59]:
```
a = 50
b = 200
if b > a:
    print("b is greater than a")
```

```
b is greater than a
```

Indentation

In [60]:
```
if 5 > 2:
print("Five is greater than two!")
```

```
  File "<ipython-input-60-a314491c53bb>", line 2
    print("Five is greater than two!")
    ^
IndentationError: expected an indented block
```

In [61]:
```
if 5 > 2:
    print("Five is greater than two!")
```

```
Five is greater than two!
```

elif

In [62]:
```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

```
a and b are equal
```

else

In [63]:
```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

```
a is greater than b
```

short hand if

In [64]:
```
a = 200
b = 30
if a > b: print("a is greater than b")
```

```
a is greater than b
```

short hand if .. else

In [65]:
```
a = 2
b = 330
print("A") if a > b else print("B")
```

```
B
```

While

In [66]:
```
i = 1
while i < 6:
    print(i)
    i += 1 # you have to increment or else the loop will continue forever
```

```
1
2
3
4
5
```

In [67]:
```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

```
1
2
3
```

In [68]:
```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

```
1
2
4
5
6
```

In [69]:
```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
```

For

In [70]:
```
# Here, x is a list item
mylist = ["hi", "hello", "bye", "whatever"]
for x in mylist:
    print(x)
```

```
hi
hello
bye
whatever
```

In [71]:
```
# Here, x is an iterator not a list item
mylist = ["hi", "hello", "bye", "whatever"]
for x in range(3): # 0, 1, 2
    print(mylist[x])
```

```
hi
hello
bye
```

In [72]:
```
adj = ["red", "ripe", "tasty"]
fruits = ["apple", "banana", "apple"]

for x in adj:
    for y in fruits:
        print(x, y)
```

```
red apple
red banana
red apple
ripe apple
ripe banana
ripe apple
tasty apple
tasty banana
tasty apple
```

In [73]:
```
adj = ["red", "ripe", "tasty"]
fruits = ["apple", "banana", "apple"]

for x, y in zip(adj, fruits):
    print(x, y)
```

```
red apple
ripe banana
tasty apple
```

Functions

In [74]:
```
def my_func():
    print("This is my function")
```

In [75]:
```
my_func()
```

```
This is my function
```

In [76]:
```
def my_func(name):
    print(f"This is {name}'s function")
```

In [77]:
```
my_func()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-77-db3ada79940f> in <module>
----> 1 my_func()

TypeError: my_func() missing 1 required positional argument: 'name'
```

In [78]:
```
def my_func(name="Mohamed"): # parameter: name, argument: "Mohamed"
    print(f"This is {name}'s function")
```

In [79]:
```
my_func()
```

```
This is Mohamed's function
```

In [80]:
```
def my_func(name, age):
    print(f"This is {name} and I'm {age} years old")
```

In [81]:
```
my_func(23, "Mohamed")
```

```
This is 23 and I'm Mohamed years old
```

In [82]:
```
my_func(age = 23, name = "Mohamed") # kwargs
```

```
This is Mohamed and I'm 23 years old
```

In [83]:
```
# return
def my_func(x,y):
    return x*y

my_func(4,5)
```

Out[83]: 20

Global vs Local variables

In [84]:
```
x = "world" # Global

def myfunc():
    x = "everyone" # Local
    print("Hi " + x)

myfunc()

print("Hi " + x)
```

```
Hi everyone
Hi world
```

In [85]:
```
x = "world" # Global

def myfunc():
    global x
    x = "everyone" # Global
    print("Hi " + x)

myfunc()

print("Hi " + x)
```

```
Hi everyone
Hi everyone
```

doc string

In [86]:
```
intro = """
hi my name
is Mohamed
how are you?
"""
```

In [87]:
```
print(intro)
```

```
hi my name
is Mohamed
how are you?
```

In [88]:
```
# adding a doc string to define the function, its parameters and its return
s for future users
def add_2_numbers(a, b = 1):
    """
    This function add 2 input number `a` and `b` and assign the result to v
ariable `c`.

    Parameters:
        a : first input number with no default value
        b : second input number with a default value of 1

    Returns:
        c : the sum of `a` and `b`

    """
    c = a + b
    return c
```

In [89]:
```
add_2_numbers(5, 6)
```

Out[89]:
```
11
```

Lambda

In [90]:
```
f = lambda x : x + 10
print(f(5))
```

```
15
```

In [91]:
```
f = lambda x, y : x + 10 * y
print(f(5, 2))
```

```
25
```

In [92]:
```
def myfunc(n):
    return lambda a : a * n

mydoubler = myfunc(2) # -> lambda a : a * 2
mytripler = myfunc(3) # -> lambda a : a * 3

print(mydoubler(11))
print(mytripler(11))
```

```
22
33
```

Classes and objects

In [93]:
```
class MyClass:
    x = 5
```

In [94]:
```
p1 = MyClass()
print(p1.x)
```

```
5
```

In [95]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

```
John
36
```

In [96]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

In [97]:
```
p1.age = 40
print(p1.age)
```

```
40
```

NumPy

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

In [98]:
```
import numpy as np # commonly used alias
```

In [99]:
```
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)
```

```
[1 2 3 4 5]
```

In [100]:
```
type(arr1)
```

Out[100]:
```
numpy.ndarray
```

In [101]:
```
arr2 = np.array([[1, 2, 3], [4, 5, 6]]) # list of lists
print(arr2)
```

```
[[1 2 3]
 [4 5 6]]
```

In [102]:
```
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr3)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

In [103]:
```
print("Array 1: ", arr1.ndim)
print("Array 2: ", arr2.ndim)
print("Array 3: ", arr3.ndim)
```

```
Array 1:  1
Array 2:  2
Array 3:  3
```

In [104]:
```
arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
```

```
[[[[[1 2 3 4]]]]]
```

In [105]:
```
arr1[0]
```

Out[105]:
```
1
```

In [106]:
```
arr2[0]
```

Out[106]:
```
array([1, 2, 3])
```

In [107]:
```
arr2[0,1]
```

Out[107]:
```
2
```

In [108]:
```
arr[0,0,0,0,1]
```

Out[108]:
```
2
```

In [109]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

```
[2 4]
```

In [110]:
```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

```
[7 8 9]
```

In [111]:
```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 1:4])
```

```
[[2 3 4]
 [7 8 9]]
```

In [112]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
arr.dtype
```

Out[112]:
```
dtype('int32')
```

In [113]:
```
print(arr.astype('f'))
```

```
[1. 2. 3. 4. 5. 6. 7.]
```

In [114]:
```
arr = np.array([1, 2, 3, 4], dtype='f')
print(arr)
```

```
[1. 2. 3. 4.]
```

In [115]:
```
arr2.shape
```

Out[115]:
```
(2, 3)
```

In [116]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr2 = arr.reshape(4, 3)
newarr3 = arr.reshape(2, 3, 2)
print(newarr2)
print("-------------")
print(newarr3)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
-------------
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

In [117]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(3, 3)
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-117-cf832910fe8f> in <module>
      1 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
----> 2 newarr = arr.reshape(3, 3)

ValueError: cannot reshape array of size 12 into shape (3,3)
```

In [118]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1) # flatten
print(newarr)
```

```
[1 2 3 4 5 6]
```

In [119]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
    print(x)
```

```
[1 2 3]
[4 5 6]
```

In [120]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
    for y in x:
        print(y)
```

```
1
2
3
4
5
6
```

In [121]:
```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
    print(x)
```

```
1
2
3
4
5
6
7
8
```

In [122]:
```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

```
[1 2 3 4 5 6]
```

In [123]:
```
arr1 + arr2
```

Out[123]:
```
array([5, 7, 9])
```

In [124]:
```
arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr)
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

In [125]:
```
arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)

print(x)
```

```
(array([3, 5, 6], dtype=int64),)
```

In [126]:
```
arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

```
['apple' 'banana' 'cherry']
```

In [127]:
```
arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

```
[[2 3 4]
 [0 1 5]]
```

In [128]:
```
arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

```
[41 43]
```

In [129]:
```
arr = np.array([41, 42, 43, 44])

filter_arr = []

for element in arr:
    if element > 42:
        filter_arr.append(True)
    else:
        filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

```
[False, False, True, True]
[43 44]
```

Pandas

In [130]:
```
import pandas as pd
```

```
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\computation\expressions.py:20: UserWarning: Pandas requires version '2.7.3' or newer of 'numexpr' (version '2.7.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
```

In [131]:
```
a = [1, 7, 2]

myser = pd.Series(a)

print(myser)
myser[1]
```

```
0    1
1    7
2    2
dtype: int64
```

Out[131]:
```
7
```

In [132]:
```
a = [1, 7, 2]

myser = pd.Series(a, index = ['x','y','z' ])

print(myser)
myser["y"]
```

```
x    1
y    7
z    2
dtype: int64
```

Out[132]:
```
7
```

In [133]:
```
calories = {"day1": 420, "day2": 380, "day3": 390}

myser = pd.Series(calories)

print(myser)
```

```
day1    420
day2    380
day3    390
dtype: int64
```

In [134]:
```
mydataset = {
  'students': ["Mohamed", "Ahmed", "Mahmoud"],
  'gpa': [4, 2.5, 3]
}

df = pd.DataFrame(mydataset)

print(df)
```

```
  students  gpa
0  Mohamed  4.0
1    Ahmed  2.5
2  Mahmoud  3.0
```

In [135]:
```
df.loc[0]
```

Out[135]:
```
students    Mohamed
gpa             4.0
Name: 0, dtype: object
```

In [136]:
```
mydataset = {
  'students': ["Mohamed", "Ahmed", "Mahmoud"],
  'gpa': [4, 2.5, 3]
}

df = pd.DataFrame(mydataset, index = ["s1","s2","s3"])

print(df)
```

```
   students  gpa
s1  Mohamed  4.0
s2    Ahmed  2.5
s3  Mahmoud  3.0
```

In [137]:
```
df.loc[0]
```

```
---------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self,
key)
   3652         try:
-> 3653             return self._engine.get_loc(casted_key)
   3654         except KeyError as err:

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.inde
x.IndexEngine.get_loc()

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.inde
x.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

KeyError: 0

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call las
t)
<ipython-input-137-7eaf75073732> in <module>
----> 1 df.loc[0]

~\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self,
key)
   1101
   1102             maybe_callable = com.apply_if_callable(key, self.obj)
-> 1103             return self._getitem_axis(maybe_callable, axis=axis)
   1104
   1105     def _is_scalar_access(self, key: tuple):

~\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(sel
f, key, axis)
   1341         # fall thru to straight lookup
   1342         self._validate_key(key, axis)
-> 1343         return self._get_label(key, axis=axis)
   1344
   1345     def _get_slice_axis(self, slice_obj: slice, axis: AxisInt):

~\anaconda3\lib\site-packages\pandas\core\indexing.py in _get_label(self,
label, axis)
   1291     def _get_label(self, label, axis: AxisInt):
   1292         # GH#5567 this will fail if the label is not present in th
e axis.
-> 1293         return self.obj.xs(label, axis=axis)
   1294
   1295     def _handle_lowerdim_multi_index_axis0(self, tup: tuple):

~\anaconda3\lib\site-packages\pandas\core\generic.py in xs(self, key, axi
s, level, drop_level)
   4093                     new_index = index[loc]
   4094         else:
-> 4095             loc = index.get_loc(key)
```

```
   4096
   4097             if isinstance(loc, np.ndarray):

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self,
key)
   3653             return self._engine.get_loc(casted_key)
   3654         except KeyError as err:
-> 3655             raise KeyError(key) from err
   3656         except TypeError:
   3657             # If we have a listlike key, _check_indexing_error wil
l raise

KeyError: 0
```

In [138]:
```
df.loc["s1"]
```

Out[138]:
```
students    Mohamed
gpa             4.0
Name: s1, dtype: object
```

In [139]:
```
df.iloc[0]
```

Out[139]:
```
students    Mohamed
gpa             4.0
Name: s1, dtype: object
```

In [140]:
```
df = pd.read_csv('Iris.csv')
```

In [141]:
```
df.head()
```

Out[141]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [142]:
```
df.tail()
```

Out[142]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [143]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [144]:
```
df.dropna(inplace=True)
```

In [145]:
```
x = df["PetalWidthCm"].mean()

df["PetalWidthCm"].fillna(x, inplace = True)
```

In [146]:
```
df.drop_duplicates(inplace = True)
```

In [147]:
```
df.drop(columns=['Id','Species']).corr()
```

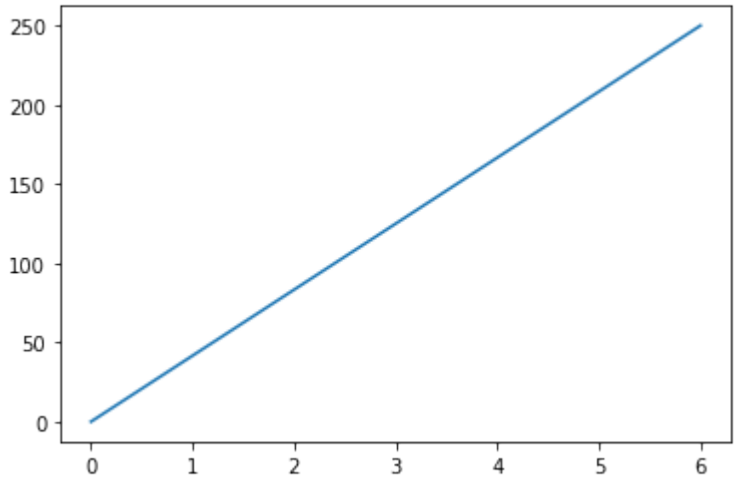
Out[147]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

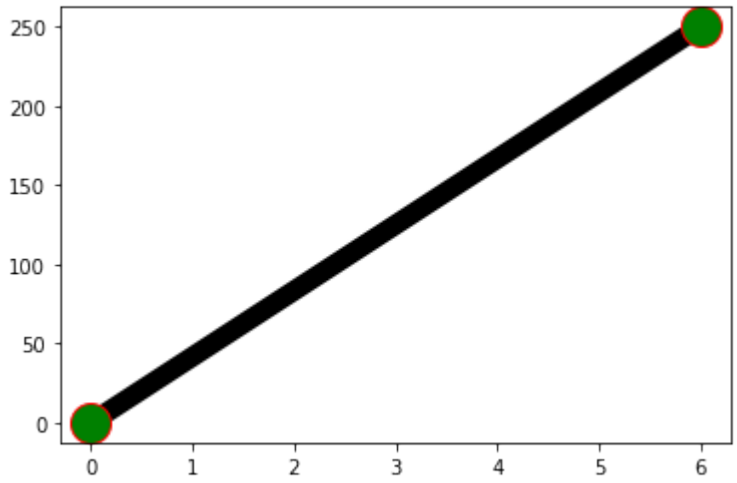
Matplotlib

In [148]:
```
import matplotlib.pyplot as plt
```

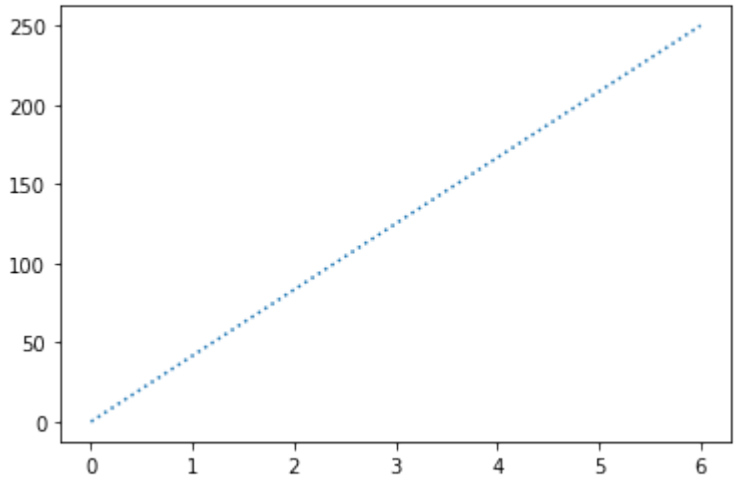
In [149]:
```
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```



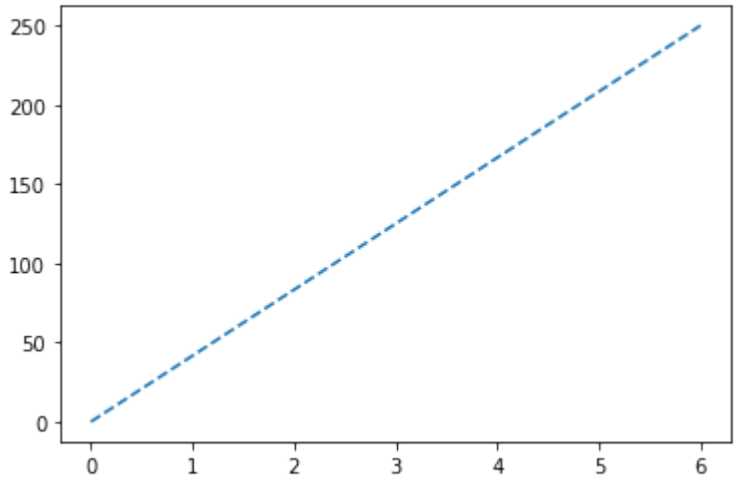
In [150]:
```
# change the design
plt.plot(xpoints, ypoints, marker = 'o', c = 'black', linewidth = '10', ms
= 20, mec = 'red', mfc = 'green')
plt.show()
```



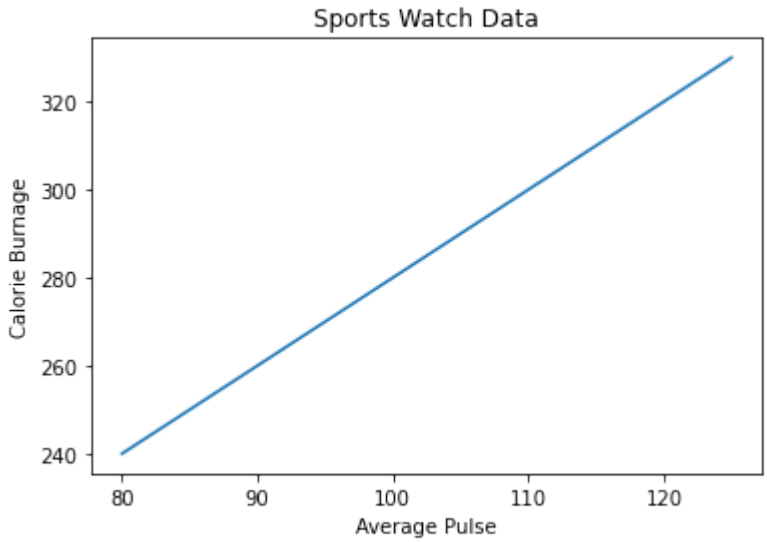
In [151]:
```
# change the style of the line
plt.plot(xpoints, ypoints, linestyle = 'dotted')
plt.show()
```



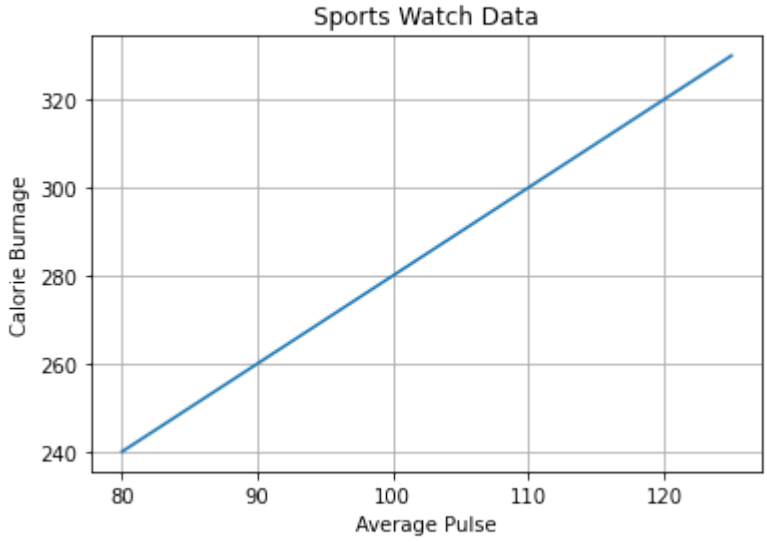
In [152]:
```
plt.plot(xpoints, ypoints, linestyle = 'dashed')
plt.show()
```



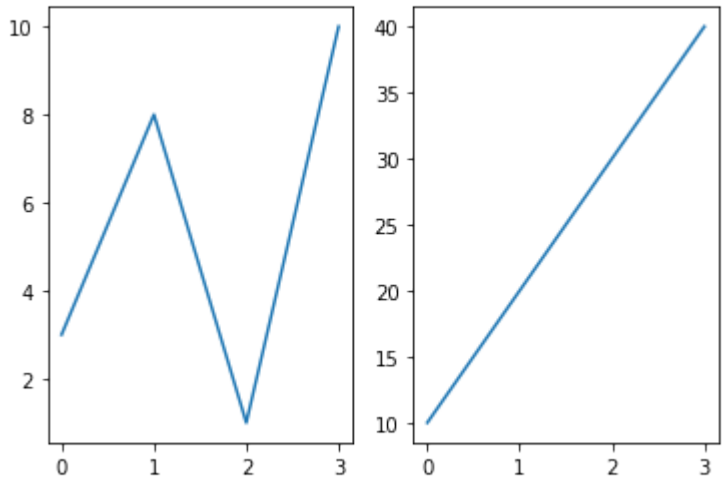
In [153]:
```
# adding a title and labels to the axes
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



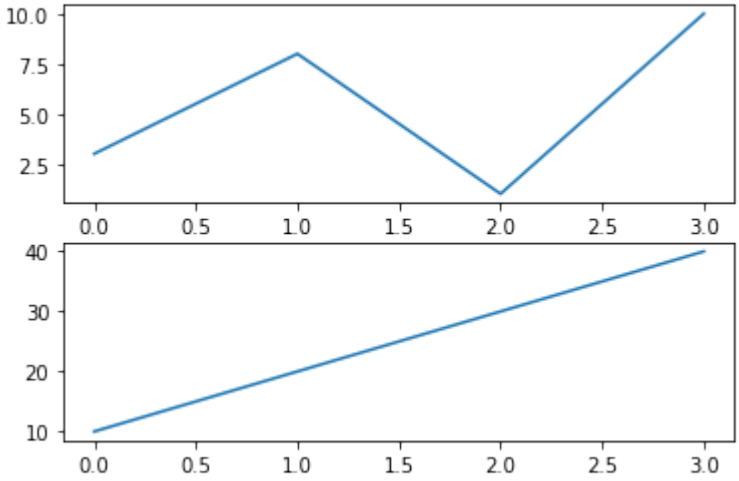
In [154]:
```
# add a grid
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.grid()
plt.show()
```



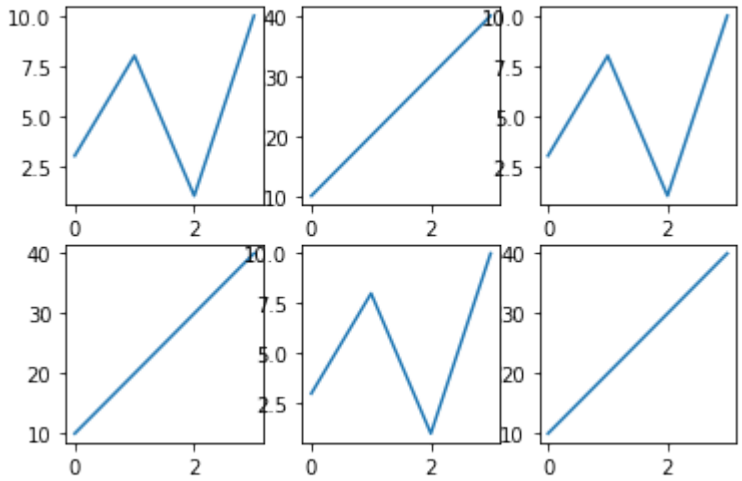
In [155]:
```
# create a plot of subplots (horizontally)
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```



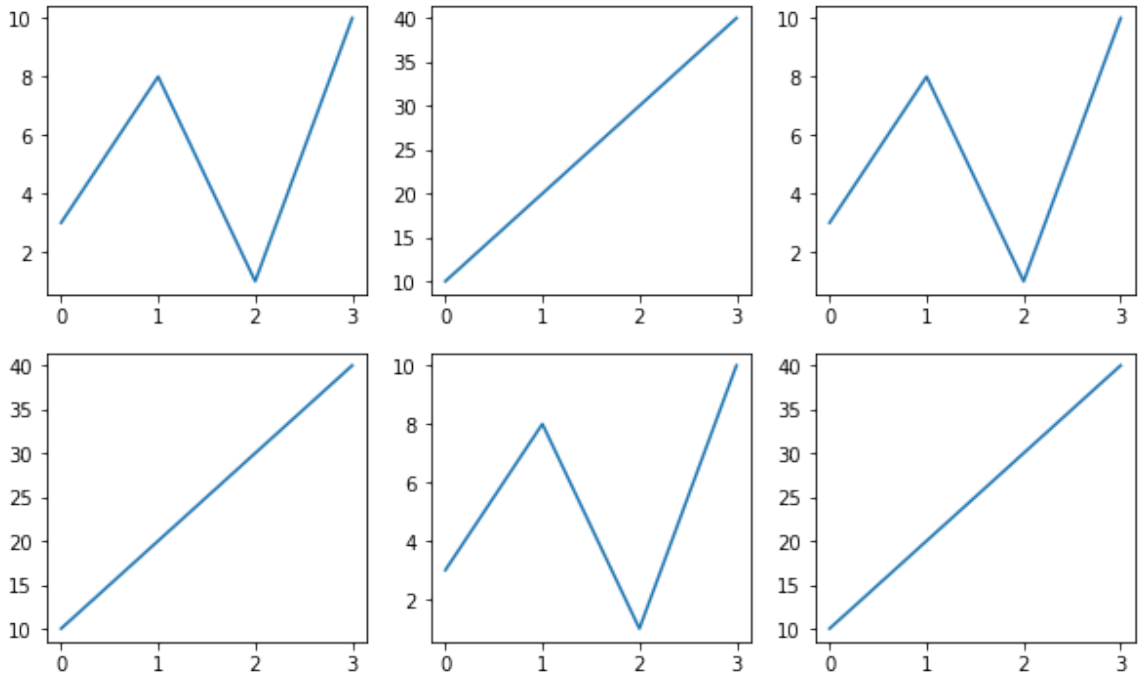
In [156]:
```
# create a plot of subplots (vertically)
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```



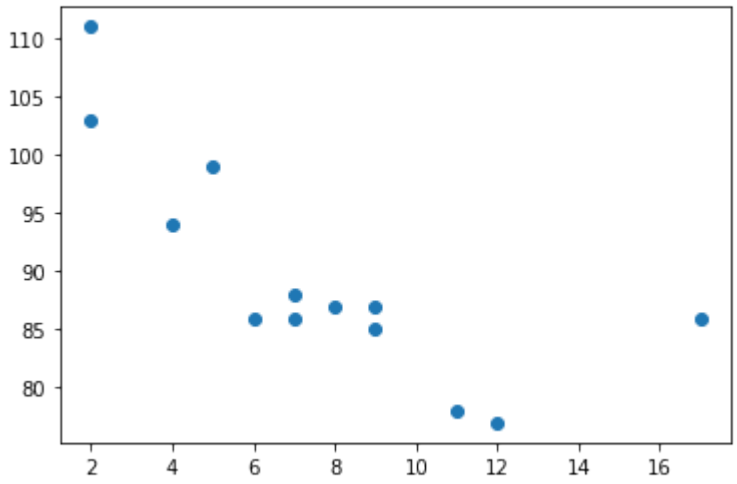
In [157]:
```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```



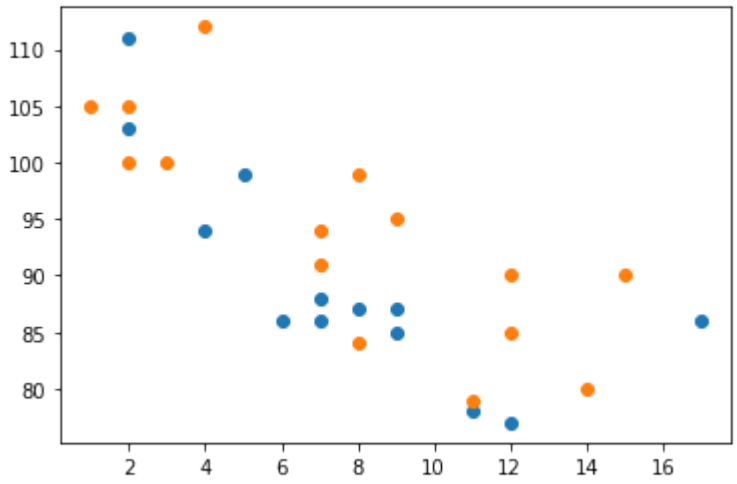
In [158]:
```
# change figure size
plt.figure(figsize=(10,6))

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```



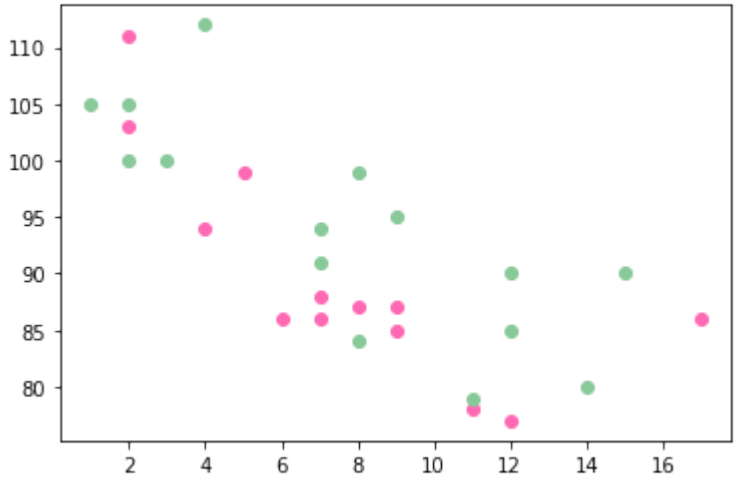
In [159]:
```
# create a scatter plot
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



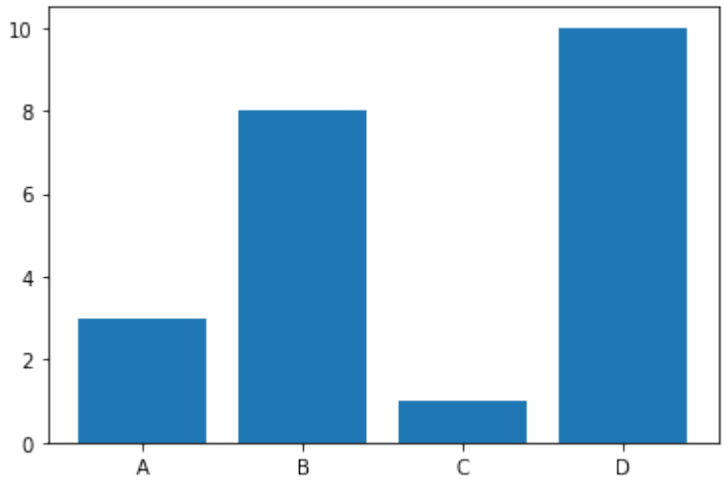
In [160]:
```
# create two plots on the same figure

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```



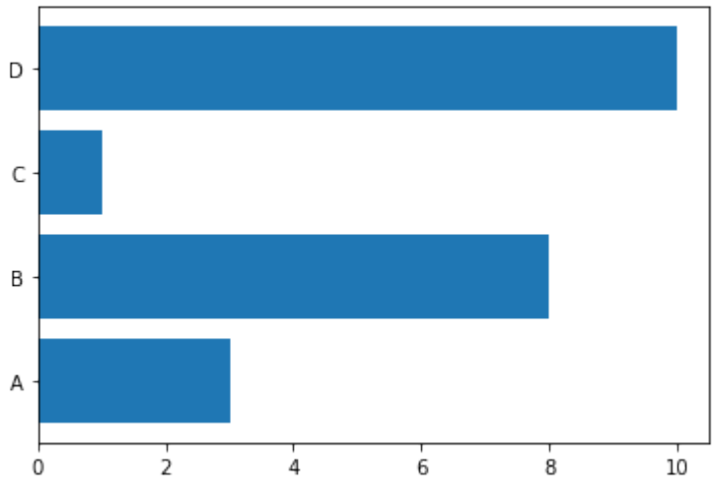
In [161]:
```
# change colors

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')

plt.show()
```



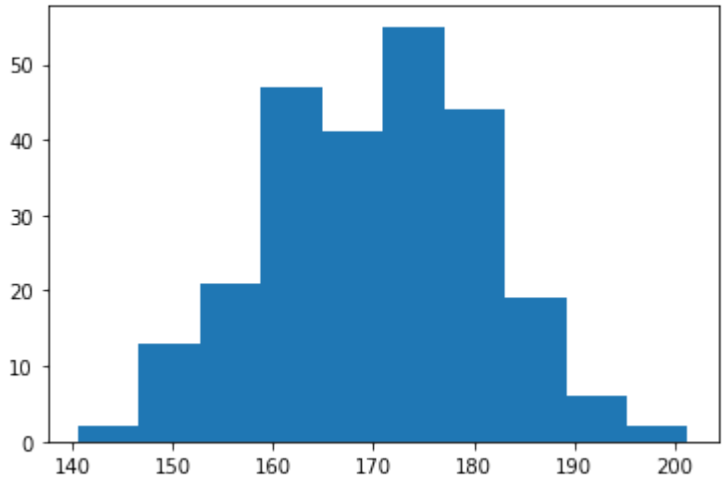
In [162]:
```
# barplot

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```



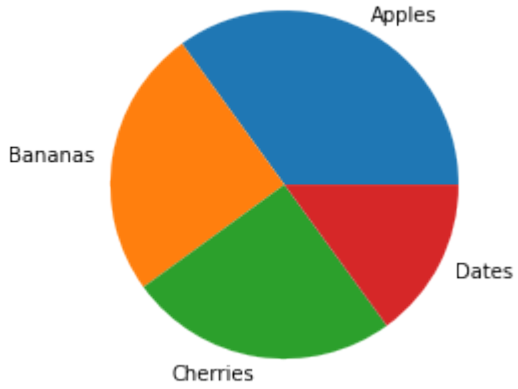
In [163]:
```
# horizontal barplot

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```



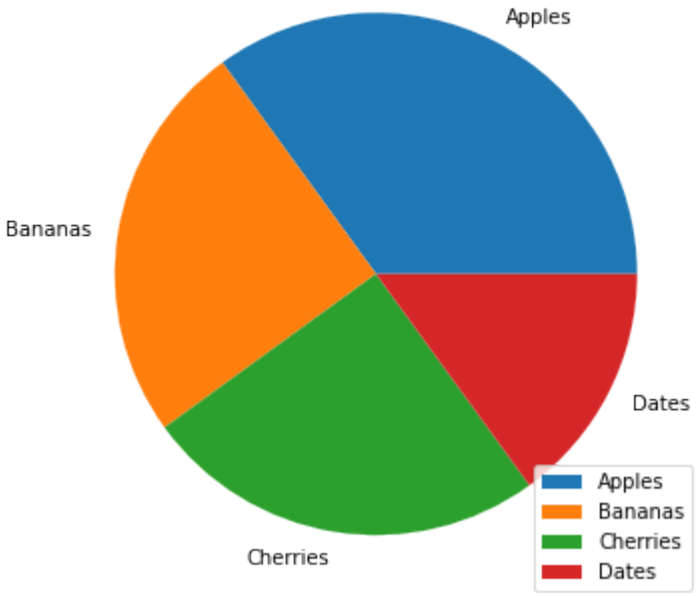
In [164]:
```
# histogram
x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```



In [165]:
```
# pie chart
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```



In [166]:
```
# adding legend
plt.figure(figsize=(10,6))
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)

plt.legend()
plt.show()
```



In [ ]: