Review on Python

Dr. Mahmoud Nabil Mahmoud
mnmahmoud@ncat.edu

North Carolina A & T State
University

September 6, 2021

Agenda

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

Outline

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

Outline

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.

Python Syntax compared to other programming languages

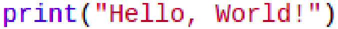
- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Hello World

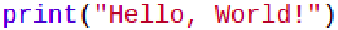
Let's write our first Python file, called helloworld.py, which can be done in any text editor.



```
print("Hello, World!")
```

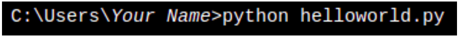
Hello World

Let’s write our first Python file, called helloworld.py, which can be done in any text editor.



```
print("Hello, World!")
```

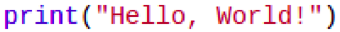
Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:



```
C:\Users\Your Name>python helloworld.py
```

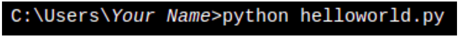
Hello World

Let's write our first Python file, called helloworld.py, which can be done in any text editor.



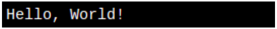
```
print("Hello, World!")
```

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:



```
C:\Users\Your Name>python helloworld.py
```

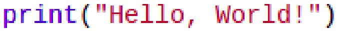
The output should read:



```
Hello, World!
```

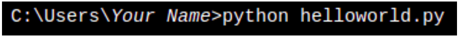
Hello World

Let's write our first Python file, called helloworld.py, which can be done in any text editor.



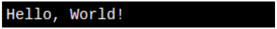
```
print("Hello, World!")
```

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:



```
C:\Users\Your Name>python helloworld.py
```

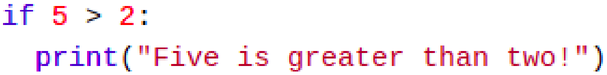
The output should read:



```
Hello, World!
```

Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- In other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code
- The number of spaces is up to you as a programmer, but it has to be at least one.



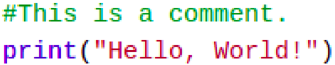
```
if 5 > 2:
  print("Five is greater than two!")
```

Outline

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

Comments

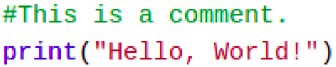
- Python has commenting capability for the purpose of in-code documentation.
- Comments start with a #, and Python will render the rest of the line as a comment:



```
#This is a comment.
print("Hello, World!")
```

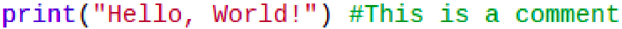
Comments

- Python has commenting capability for the purpose of in-code documentation.
- Comments start with a #, and Python will render the rest of the line as a comment:



```
#This is a comment.
print("Hello, World!")
```

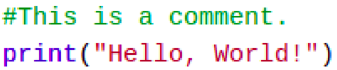
- Comments can be placed at the end of a line, and Python will ignore the rest of the line:



```
print("Hello, World!") #This is a comment
```

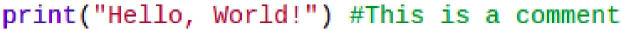
Comments

- Python has commenting capability for the purpose of in-code documentation.
- Comments start with a #, and Python will render the rest of the line as a comment:



```
#This is a comment.
print("Hello, World!")
```

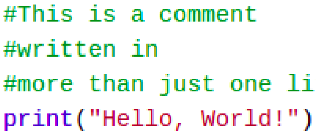
- Comments can be placed at the end of a line, and Python will ignore the rest of the line:



```
print("Hello, World!") #This is a comment
```

Multi Line Comments

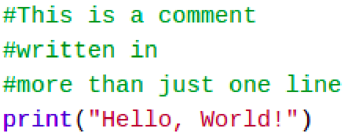
- To add a multiline comment you could insert a # for each line:



```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

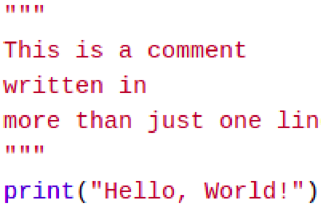
Multi Line Comments

- To add a multiline comment you could insert a # for each line:



```python
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

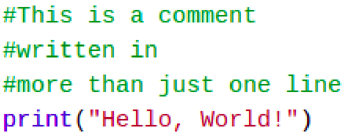
- You can also add a multiline string (triple quotes) as a comment



```python
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

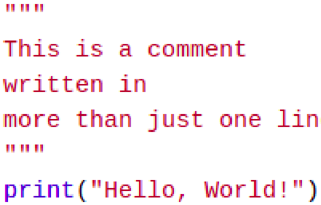
Multi Line Comments

- To add a multiline comment you could insert a # for each line:



```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

- You can also add a multiline string (triple quotes) as a comment



```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Outline

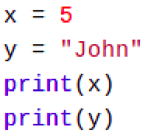
1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

Python Variables

Variables

Variables are containers for storing data values.

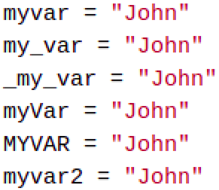
- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.
- Variables do not need to be declared with any particular type, and can even change type after they have been set.



```
x = 5
y = "John"
print(x)
print(y)
```

Variable Names

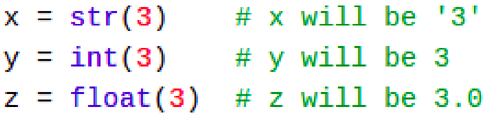
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)



```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Casting and Type

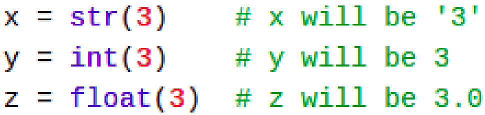
If you want to specify the data type of a variable, this can be done with casting.



```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

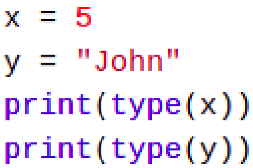
Casting and Type

If you want to specify the data type of a variable, this can be done with casting.



```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

You can get the data type of a variable with the type() function.



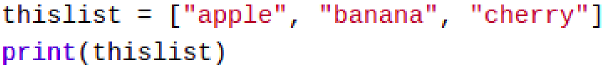
```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Outline

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

Python Lists

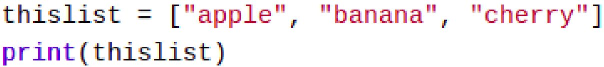
Lists are used to store multiple items in a single variable.



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

Python Lists

Lists are used to store multiple items in a single variable.



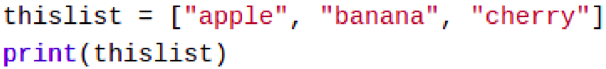
```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List items are changeable.

thislist[2]=”mango”

Python Lists

Lists are used to store multiple items in a single variable.

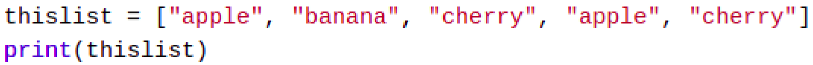


```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List items are changeable.

thislist[2]=”mango”

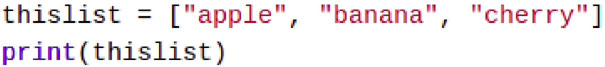
List items are allow duplicates.



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

Python Lists

Lists are used to store multiple items in a single variable.

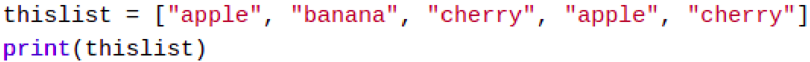


```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List items are changeable.

thislist[2]=”mango”

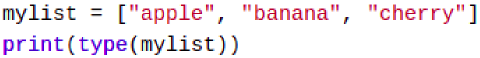
List items are allow duplicates.



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

Python Lists

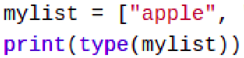
From Python's perspective, lists are defined as objects with the data type 'list':



```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

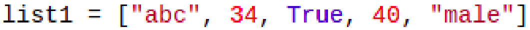
Python Lists

From Python's perspective, lists are defined as objects with the data type 'list':



```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

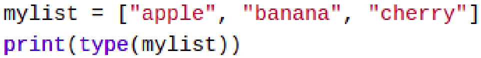
A list can contain different data types:



```
list1 = ["abc", 34, True, 40, "male"]
```

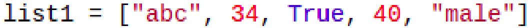
Python Lists

From Python's perspective, lists are defined as objects with the data type 'list':



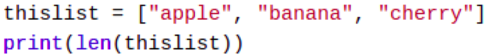
```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

A list can contain different data types:



```
list1 = ["abc", 34, True, 40, "male"]
```

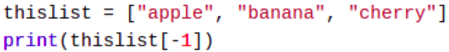
To determine how many items a list has, use the len() function:



```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Python Lists

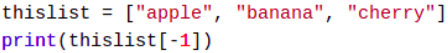
Negative indexing means start from the end



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

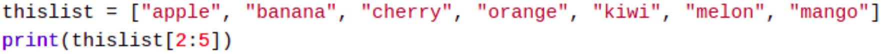
Python Lists

Negative indexing means start from the end



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

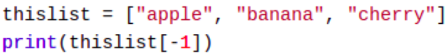
You can specify a range of indexes by specifying where to start and where to end the range.



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

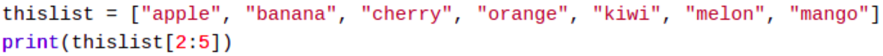
Python Lists

Negative indexing means start from the end



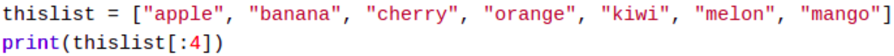
```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

You can specify a range of indexes by specifying where to start and where to end the range.



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

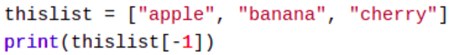
By leaving out the start value, the range will start at the first item:



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

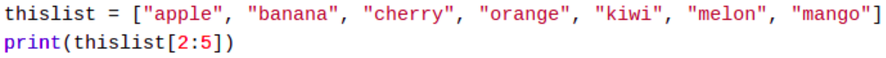
Python Lists

Negative indexing means start from the end



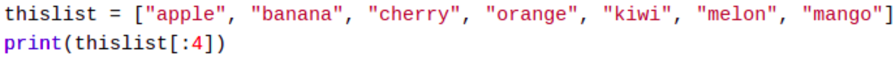
```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

You can specify a range of indexes by specifying where to start and where to end the range.



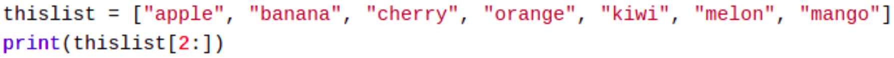
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

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



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

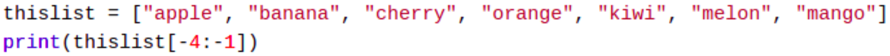
By leaving out the end value, the range will go on to the end of the list:



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

Python Lists

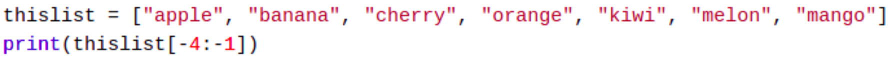
Specify negative indexes if you want to start the search from the end of the list:



```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

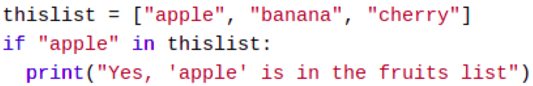
Python Lists

Specify negative indexes if you want to start the search from the end of the list:



```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

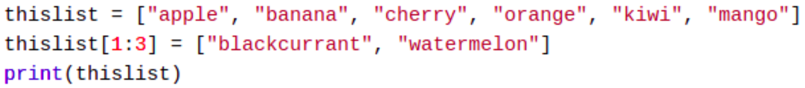
To determine if a specified item is present in a list use the in keyword:



```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Python Lists

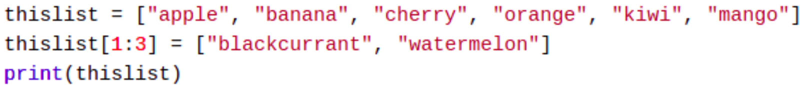
To change the value of items within a specific range.



```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

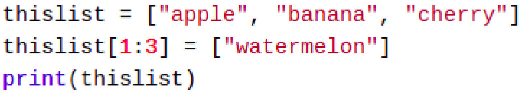
Python Lists

To change the value of items within a specific range.



```python
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

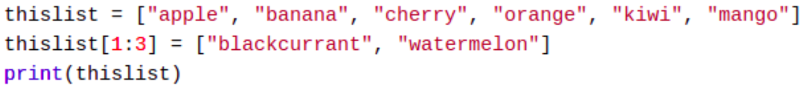
Change the second and third value by replacing it with one value:



```python
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

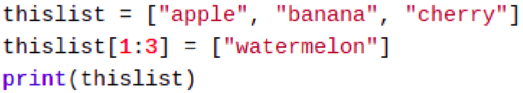
Python Lists

To change the value of items within a specific range.



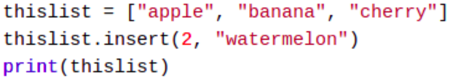
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

Change the second and third value by replacing it with one value:



```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

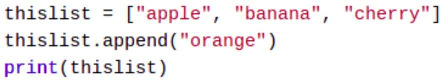
Insert "watermelon" as the third item:



```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Python Lists

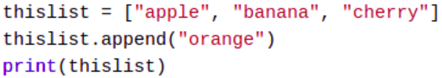
To add an item to the end of the list, use the append() method:



```python
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

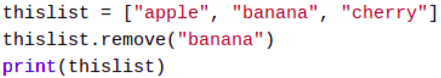
Python Lists

To add an item to the end of the list, use the append() method:



```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

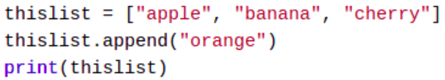
The remove() method removes the specified item.



```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

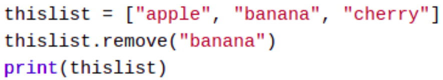
Python Lists

To add an item to the end of the list, use the append() method:



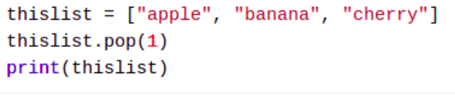
```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

The remove() method removes the specified item.



```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

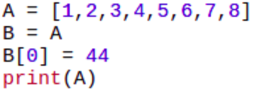
The pop() method removes the specified index.



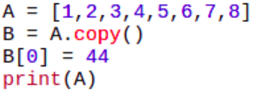
```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

Python Lists

- Lists are reference type.



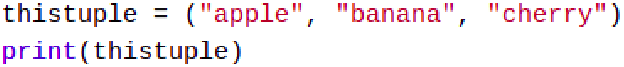
```
A = [1,2,3,4,5,6,7,8]
B = A
B[0] = 44
print(A)
```



```
A = [1,2,3,4,5,6,7,8]
B = A.copy()
B[0] = 44
print(A)
```

Tubles

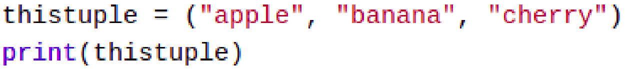
A tuple is a collection which is ordered and unchangeable.



```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

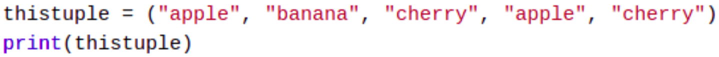
Tubles

A tuple is a collection which is ordered and unchangeable.



```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

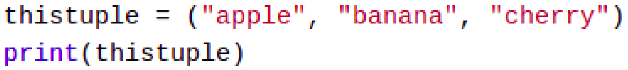
tuples allow duplicates



```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

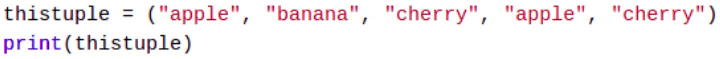
Tubles

A tuple is a collection which is ordered and unchangeable.



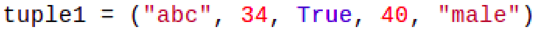
```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

tuples allow duplicates



```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

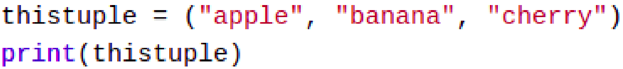
A tuple can contain different data types:



```
tuple1 = ("abc", 34, True, 40, "male")
```

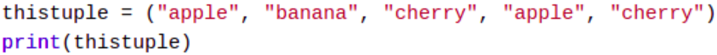
Tubles

A tuple is a collection which is ordered and unchangeable.



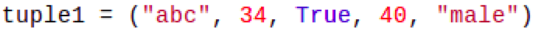
```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

tuples allow duplicates



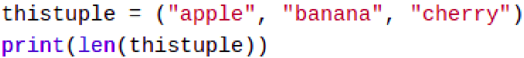
```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

A tuple can contain different data types:



```
tuple1 = ("abc", 34, True, 40, "male")
```

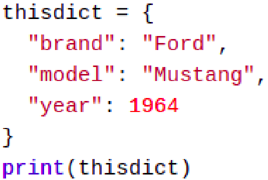
To determine how many items a tuple has, use the len() function:



```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Dictionary

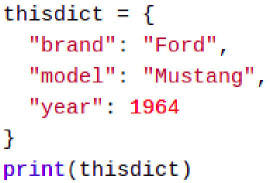
Dictionaries are used to store data values in key:value pairs.
A dictionary is a collection which is , changeable and does not allow duplicates (keys).



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

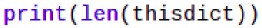
Dictionary

Dictionaries are used to store data values in key:value pairs.
A dictionary is a collection which is , changeable and does not allow duplicates (keys).



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

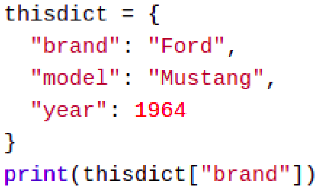
To determine how many items a dictionary has, use the len() function:



```
print(len(thisdict))
```

Dictionary

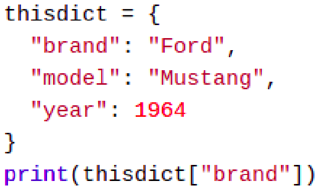
Dictionary items are presented in key:value pairs, and can be referred to by using the key name.



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

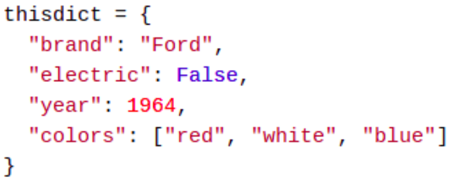
Dictionary

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

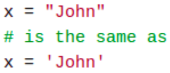
The values in dictionary items can be of any data type:



```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

Strings

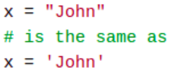
String variables can be declared either by using single or double quotes:



```
x = "John"
# is the same as
x = 'John'
```

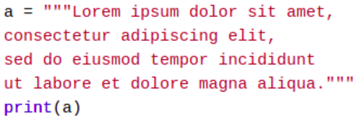
Strings

String variables can be declared either by using single or double quotes:



```
x = "John"
# is the same as
x = 'John'
```

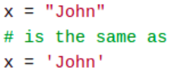
You can assign a multiline string to a variable by using three quotes:



```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

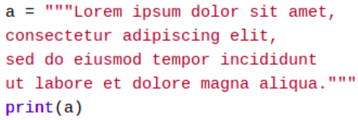
Strings

String variables can be declared either by using single or double quotes:



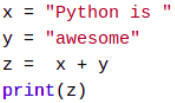
```
x = "John"
# is the same as
x = 'John'
```

You can assign a multiline string to a variable by using three quotes:



```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

You can concatenate two strings using $+$ operator



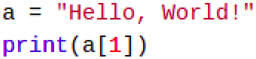
```
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

Outline

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

Strings

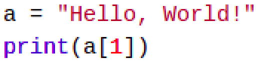
Like many other popular programming languages, strings in Python are arrays starts with index 0:



```
a = "Hello, World!"
print(a[1])
```

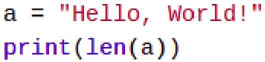
Strings

Like many other popular programming languages, strings in Python are arrays starts with index 0:



```
a = "Hello, World!"
print(a[1])
```

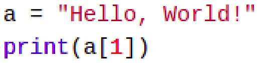
To get the length of a string, use the len() function.



```
a = "Hello, World!"
print(len(a))
```

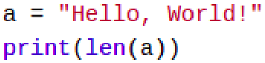
Strings

Like many other popular programming languages, strings in Python are arrays starts with index 0:



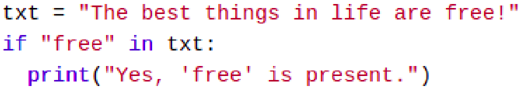
```
a = "Hello, World!"
print(a[1])
```

To get the length of a string, use the len() function.



```
a = "Hello, World!"
print(len(a))
```

To check if a certain phrase or character is present in a string, we can use the keyword in.

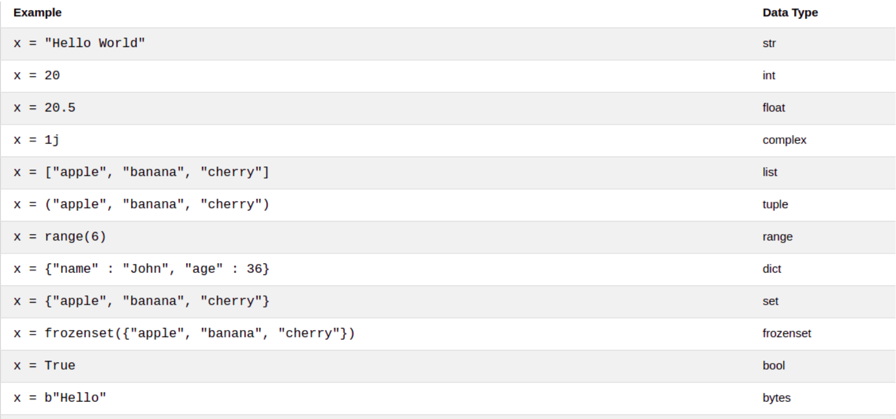


```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

Outline

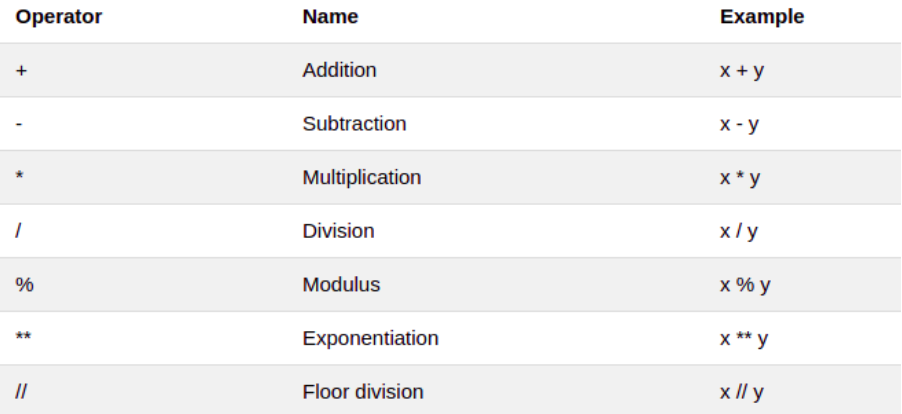
1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

Data Types



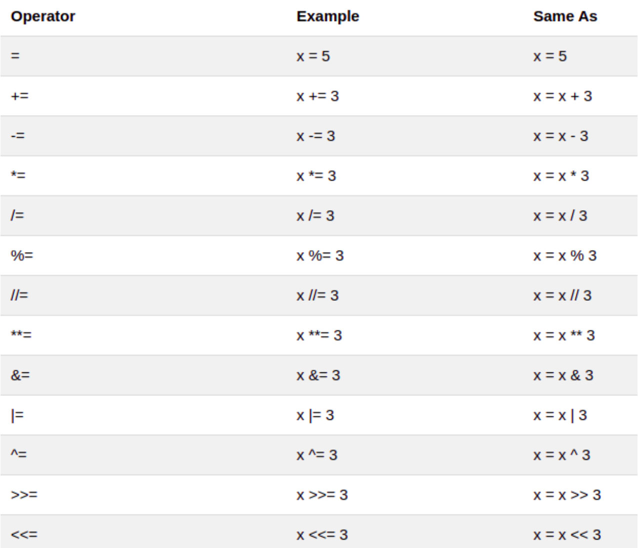
Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes

Arithmetic Operators



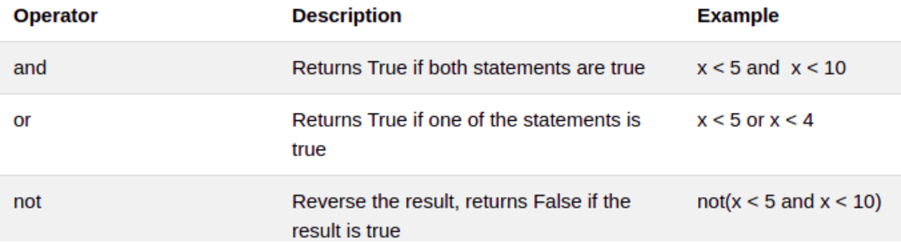
Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operators



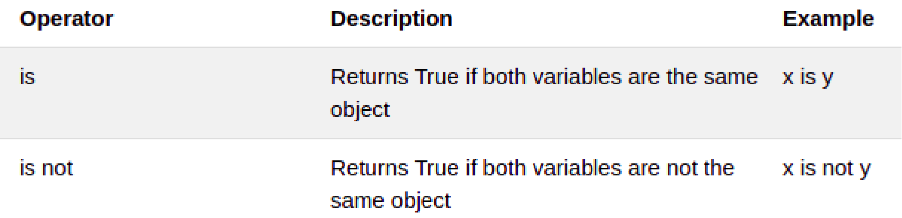
Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Logical Operators



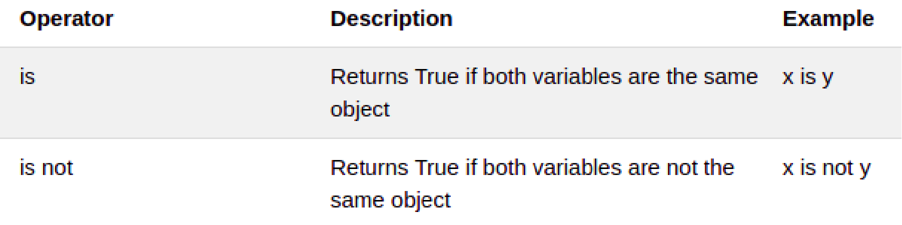
Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity and Membership Operators

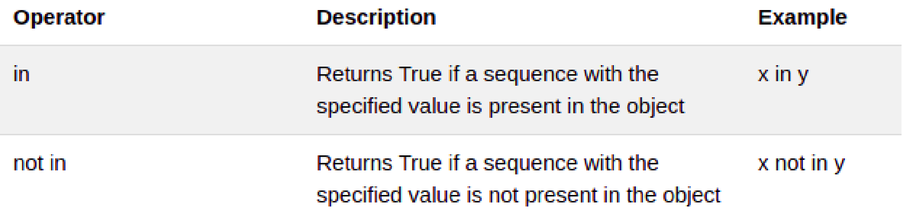


Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Identity and Membership Operators

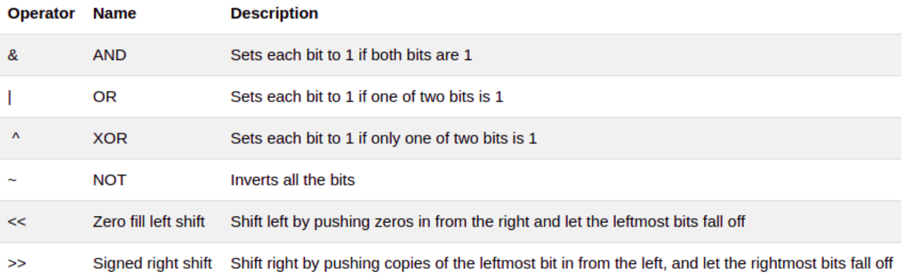


Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y



Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise Operators



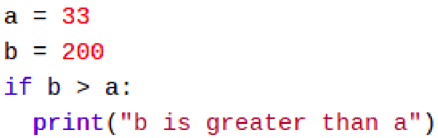
Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Outline

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

IF Statement

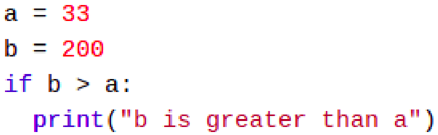
An "if statement" is written by using the if keyword.



```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

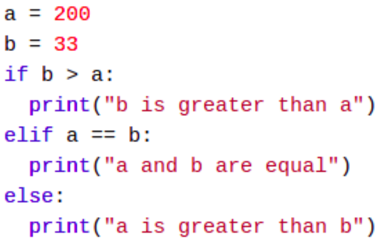
IF Statement

An "if statement" is written by using the if keyword.



```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

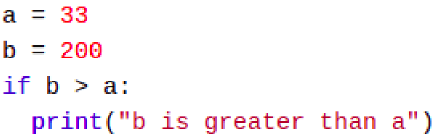
The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".



```
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")
```

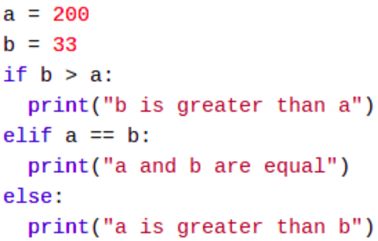
IF Statement

An "if statement" is written by using the if keyword.



```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

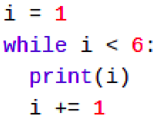
The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".



```
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")
```

While Loop

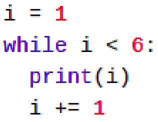
With the while loop we can execute a set of statements as long as a condition is true.



```
i = 1
while i < 6:
  print(i)
  i += 1
```

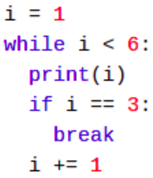
While Loop

With the while loop we can execute a set of statements as long as a condition is true.



```
i = 1
while i < 6:
  print(i)
  i += 1
```

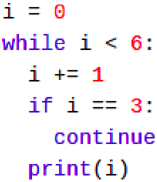
With the break statement we can stop the loop even if the while condition is true:



```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

While Loop

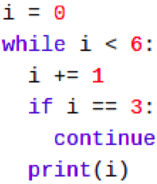
With the continue statement we can stop the current iteration, and continue with the next:



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

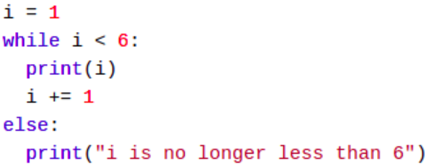
While Loop

With the continue statement we can stop the current iteration, and continue with the next:



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

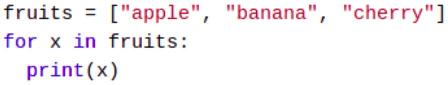
With the else statement we can run a block of code once when the condition no longer is true:



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

For Loops

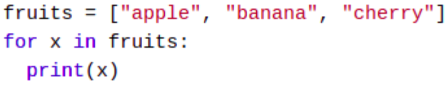
A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).



```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

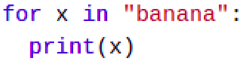
For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).



```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

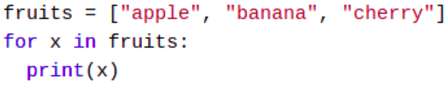
Even strings are iterable objects, they contain a sequence of characters:



```
for x in "banana":
  print(x)
```

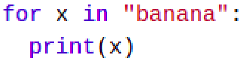
For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).



```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Even strings are iterable objects, they contain a sequence of characters:

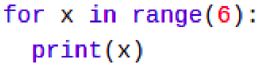


```
for x in "banana":
  print(x)
```

You can also use the continue and the break statements.

Range Function

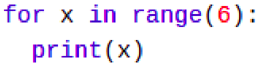
The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.



```
for x in range(6):
  print(x)
```

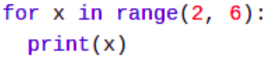
Range Function

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.



```
for x in range(6):
  print(x)
```

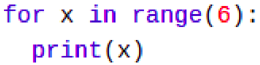
Using the start parameter:



```
for x in range(2, 6):
  print(x)
```

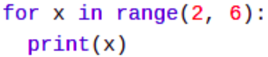
Range Function

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.



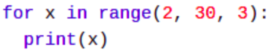
```
for x in range(6):
  print(x)
```

Using the start parameter:



```
for x in range(2, 6):
  print(x)
```

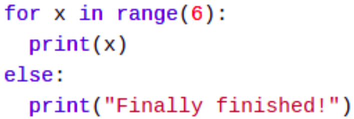
Increment the sequence with 3 (default is 1):



```
for x in range(2, 30, 3):
  print(x)
```

For Loop

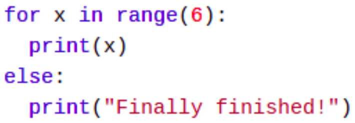
Print all numbers from 0 to 5, and print a message when the loop has ended:



```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

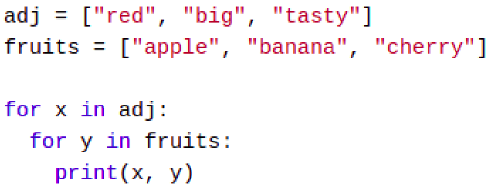
For Loop

Print all numbers from 0 to 5, and print a message when the loop has ended:



```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

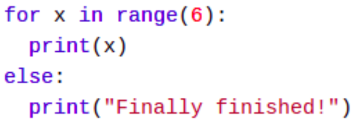
A nested loop is a loop inside a loop.



```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

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

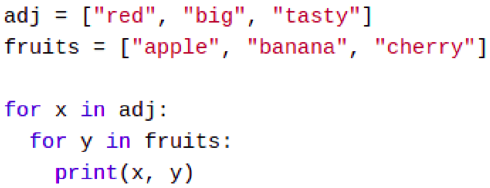
For Loop

Print all numbers from 0 to 5, and print a message when the loop has ended:



```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

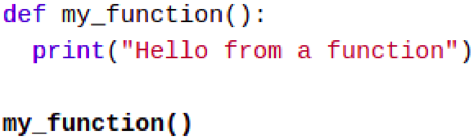
A nested loop is a loop inside a loop.



```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

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

Functions

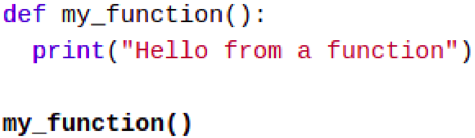
- In Python a function is defined using the def keyword:
- To call a function, use the function name followed by parenthesis:



```
def my_function():
  print("Hello from a function")

my_function()
```

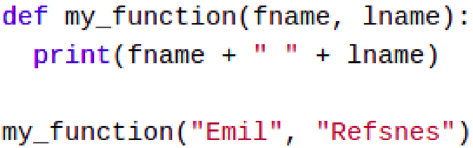
Functions

- In Python a function is defined using the def keyword:
- To call a function, use the function name followed by parenthesis:



```
def my_function():
  print("Hello from a function")

my_function()
```

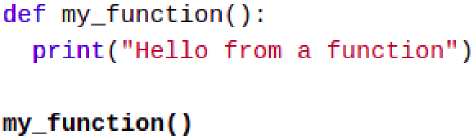
This function expects 2 arguments, and gets 2 arguments:



```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

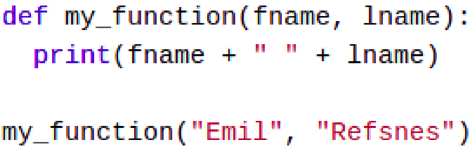
Functions

- In Python a function is defined using the def keyword:
- To call a function, use the function name followed by parenthesis:



```
def my_function():
  print("Hello from a function")

my_function()
```

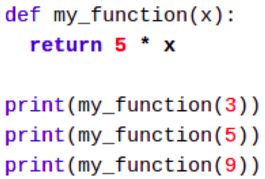
This function expects 2 arguments, and gets 2 arguments:



```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

Functions

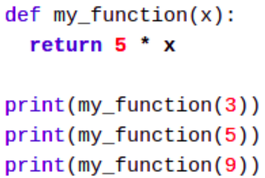
To let a function return a value, use the return statement:



```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

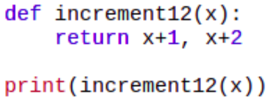
Functions

To let a function return a value, use the return statement:



```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

In python Functions can return more than one value



```
def increment12(x):
    return x+1, x+2

print(increment12(x))
```

Outline

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

NumPy

- NumPy is a Python library used for working with arrays.

NumPy

- NumPy is a Python library used for working with arrays.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

NumPy

- NumPy is a Python library used for working with arrays.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

NumPy

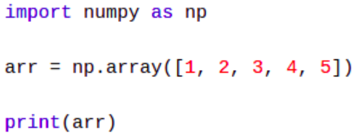
- NumPy is a Python library used for working with arrays.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

NumPy

- NumPy is a Python library used for working with arrays.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.
- Bring MATLAB power to python.

Using Numpy

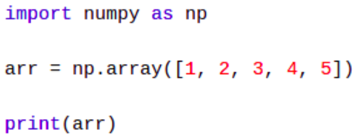
NumPy package can be referred to imported and renamed as np instead of numpy.



```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

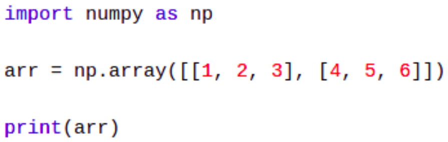
Using Numpy

NumPy package can be referred to imported and renamed as np instead of numpy.



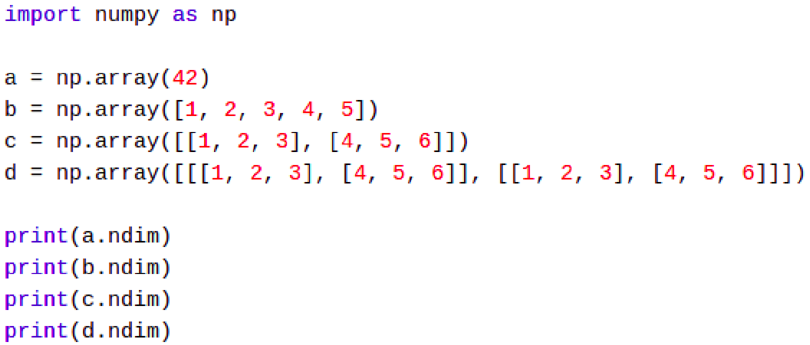
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

The array object in NumPy is called ndarray.



```
import numpy as np

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

print(arr)
```

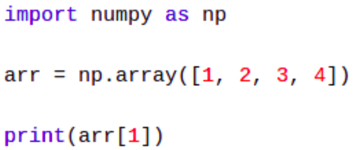
Check Number of Dimensions



```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Access Array Elements

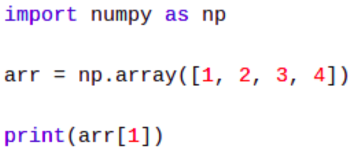
Get the second element from the following array.



```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[1])
```

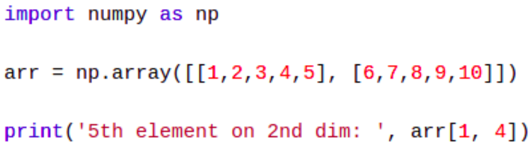
Access Array Elements

Get the second element from the following array.



```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[1])
```

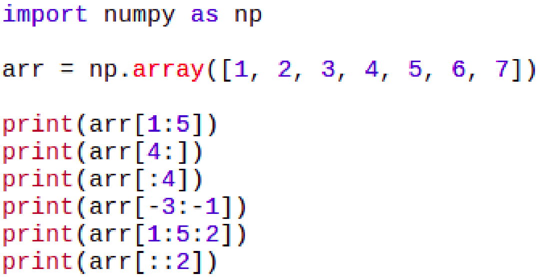
Access the 5th element on 2nd dim:



```
import numpy as np

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

print('5th element on 2nd dim: ', arr[1, 4])
```

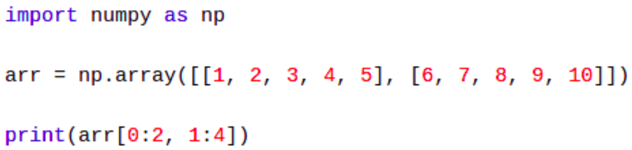
Slicing arrays

- We pass slice instead of index like this: [start:end].
- We can also define the step, like this: [start:end:step].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1



```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
print(arr[4:])
print(arr[:4])
print(arr[-3:-1])
print(arr[1:5:2])
print(arr[::2])
```

Slicing 2-D Arrays

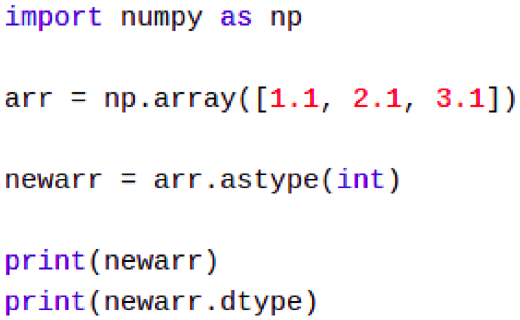


```
import numpy as np

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

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

Numpy Data Types

- NumPy has some extra data types but out of our scope.

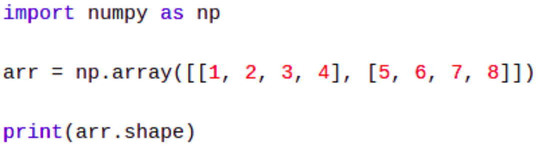
Change data type from float to integer by using int as parameter value:



```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype(int)

print(newarr)
print(newarr.dtype)
```

Array Shape

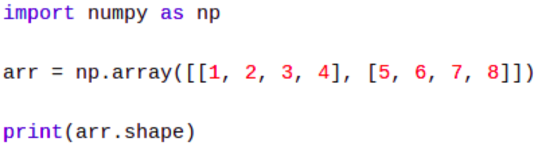
NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.



```
import numpy as np

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

print(arr.shape)
```

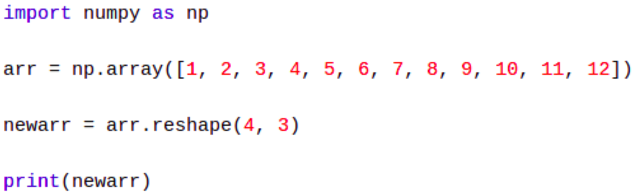
Array Shape

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.



```
import numpy as np

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

print(arr.shape)
```

Convert the following 1-D array with 12 elements into a 2-D array.

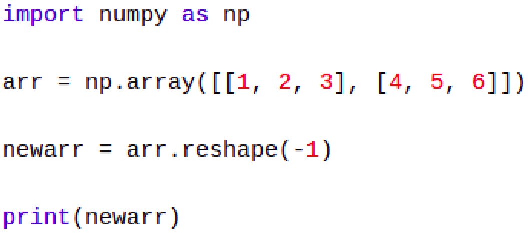


```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

Array Shape

- As long as the elements required for reshaping are equal in both shapes, you can reshape to any shape.

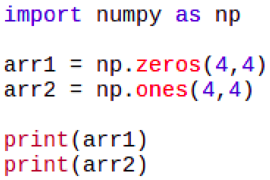
Convert the array into a 1D array:



```
import numpy as np

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

newarr = arr.reshape(-1)

print(newarr)
```

NumPy Zeros and Ones

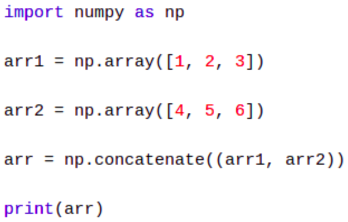
- You can initialize numpy array of all zeros or all ones.



```
import numpy as np

arr1 = np.zeros(4,4)
arr2 = np.ones(4,4)

print(arr1)
print(arr2)
```

NumPy Joining Array

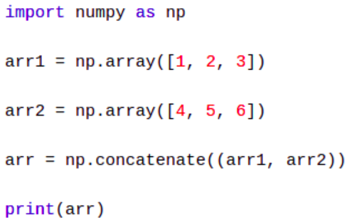
Join two arrays:



```
import numpy as np

arr1 = np.array([1, 2, 3])

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

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

print(arr)
```

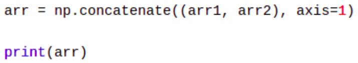
NumPy Joining Array

Join two arrays:



```
import numpy as np

arr1 = np.array([1, 2, 3])

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

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

print(arr)
```

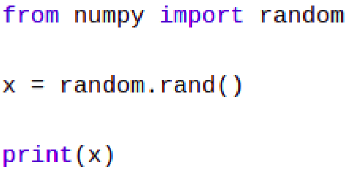
Join two 2-D arrays along rows (axis=1):



```
arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

NumPy Random

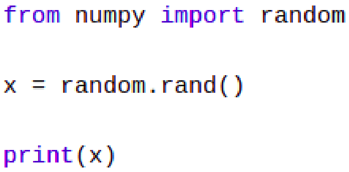
Generate a random float from 0 to 1:



```
from numpy import random

x = random.rand()

print(x)
```

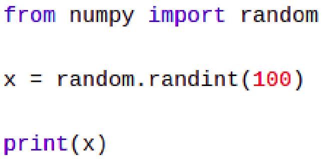
NumPy Random

Generate a random float from 0 to 1:



```
from numpy import random

x = random.rand()

print(x)
```

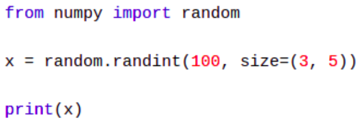
Generate a random integer from 0 to 100:



```
from numpy import random

x = random.randint(100)

print(x)
```

NumPy Random

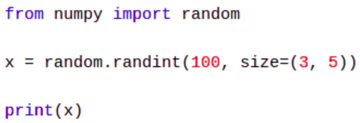
Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:



```
from numpy import random

x = random.randint(100, size=(3, 5))

print(x)
```

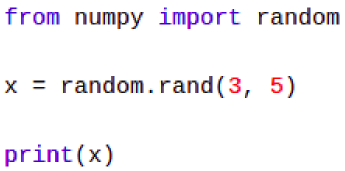
NumPy Random

Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:



```
from numpy import random

x = random.randint(100, size=(3, 5))

print(x)
```

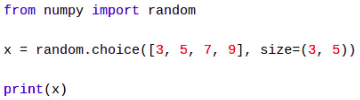
Generate a 2-D array with 3 rows, each row containing 5 random numbers:



```
from numpy import random

x = random.rand(3, 5)

print(x)
```

NumPy Random

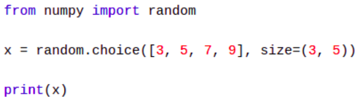
Generate a 2-D array that consists of the values in the array parameter (3, 5, 7, and 9):



```
from numpy import random

x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

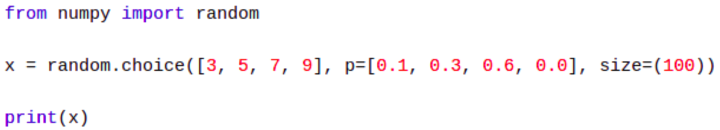
NumPy Random

Generate a 2-D array that consists of the values in the array parameter (3, 5, 7, and 9):



```
from numpy import random

x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

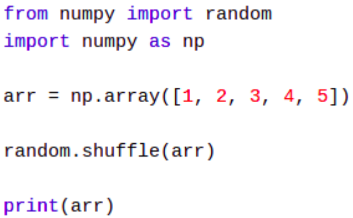
Generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9 with probability.



```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(100))

print(x)
```

NumPy Random

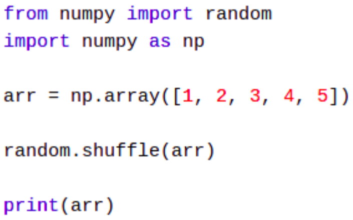
Randomly shuffle elements of following array:



```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

random.shuffle(arr)

print(arr)
```

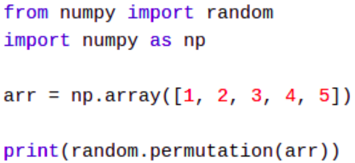
NumPy Random

Randomly shuffle elements of following array:



```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

random.shuffle(arr)

print(arr)
```

Generate a random permutation of elements of following array:



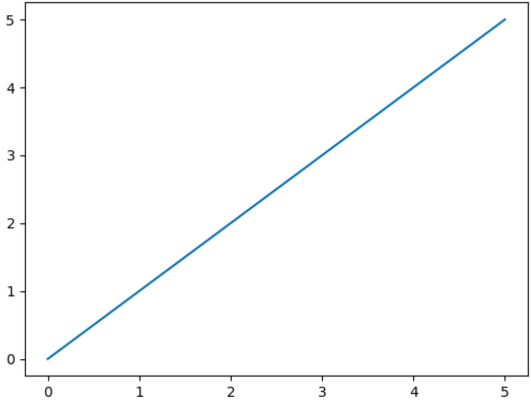
```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(random.permutation(arr))
```

Outline

1. Python Programming Language
   - Intro
   - Comments
   - Variables
   - Collections: Lists, Tuples, Dictionary
   - Strings
   - Data Types and Operators
   - Control Flow Statements
2. Numpy
3. Matplotlib

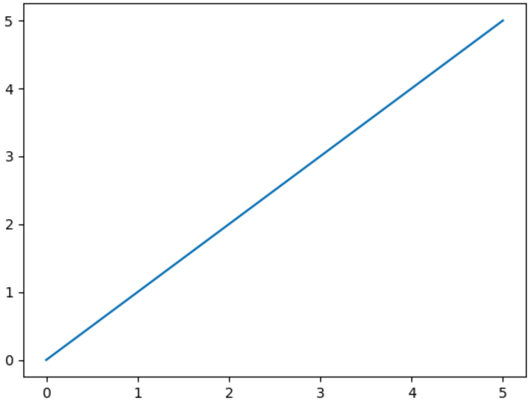
Matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.



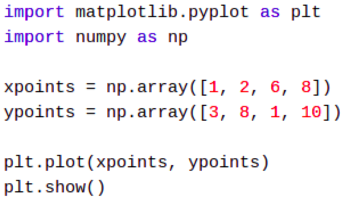
Matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.



Matplotlib

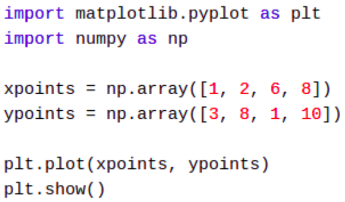
Most of the Matplotlib utilities lies under the pyplot submodule
Draw a line in a diagram from position (0,0) to position (6,250):



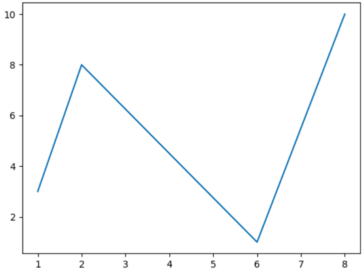
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

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

Matplotlib

Most of the Matplotlib utilities lies under the pyplot submodule
Draw a line in a diagram from position (0,0) to position (6,250):

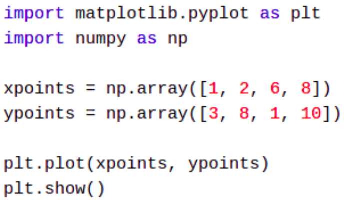


```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

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

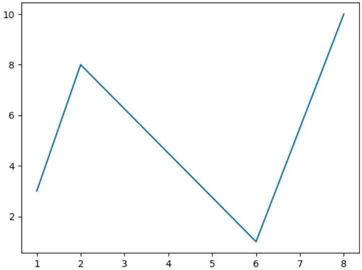


Matplotlib

Most of the Matplotlib utilities lies under the pyplot submodule
Draw a line in a diagram from position (0,0) to position (6,250):

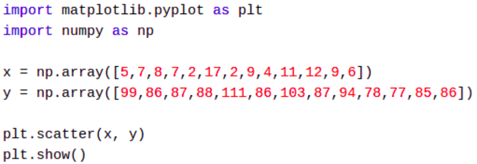


```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

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

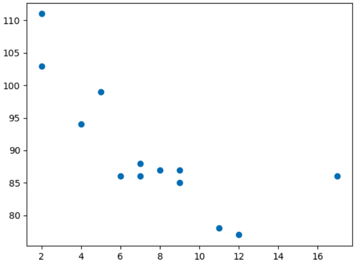


Matplotlib

A simple scatter plot:



```
import matplotlib.pyplot as plt
import numpy as np

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()
```



References

- https://www.w3schools.com/

Questions