Tutorial on Python

ENGG 818

Advanced Numerical Method

Fall 2022

Surya Pusapati

Table of content

1.1 What is Python?

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is a dynamically-typed and garbage-collected programming language. It was created by Guide van Rossum during 1985-1990. Python source code is also available under GNU General Public License (GPL).

1.2 Why learn python?

Python is consistently rated as one of the world’s most popular programming languages*.

- Python is open source which means it is available free of cost
- Python is simple and so easy to learn
- Python is versatile and can be used to create many different things
- Python has powerful development libraries, including Artificial intelligence, Machine Learning, etc.

* From source https://www.tiobe.com/tiobe-index/python/

1.3 Properties of Python

- Easy-to-learn
- Easy-to-read
- Easy-to-maintain
- A broad standard library
- Interactive Mode
- Portable
- Extendable
- Databases
- GUI Programming
- Scalable
- Object-Oriented Programming
- Beginner's Language

1.4 Python is primarily used by:

- Game Developer
- Web Designer
- Python Developer
- Full-stack Developer
- Machine Learning Engineer
- Data Scientist
- Data Analyst
- Data Engineer
- DevOps Engineer
- Software Engineer
- Researcher
- Professor

1.5 Python installation

- Python is available for download at https://www.python.org/
- Well-known Python IDE software:
  - Atom
  - IDLE
  - Jupyter Notebook
  - PyCharm
  - PyDev
  - Spyder
  - Thonny
  - Visual Studio Code

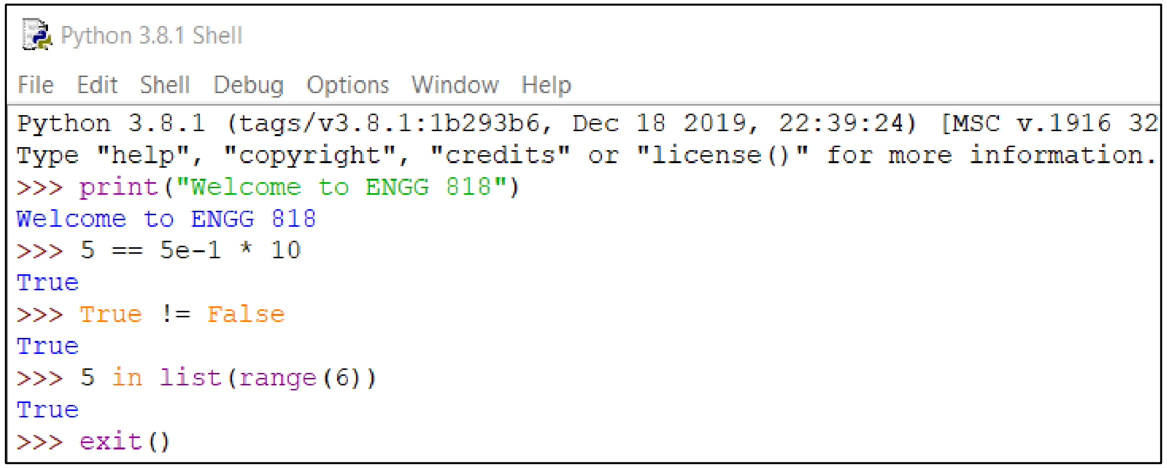
2. Introduction to python

1. Mode of Programming
2. Python Identifiers
3. Reserved Words
4. Lines and Indentation
5. Variable Assignment
6. Variable Types
7. List of built-in Methods
8. Basic Operators
9. Conditional Operators
10. Loop Statements
11. Files I/O
12. Functions
13. Python Modules

2.1 Mode of Programming

Interactive Mode Programming

- A prompt with an active python interpreter without passing a script file as a parameter.

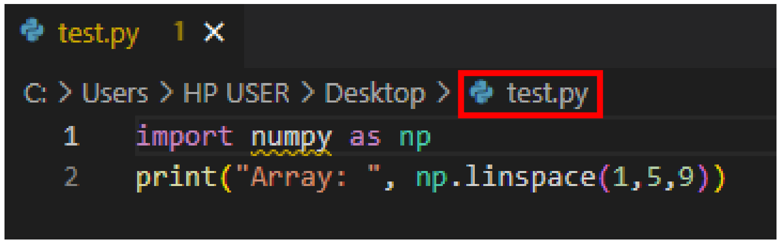


```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Welcome to ENGG 818")
Welcome to ENGG 818
>>> 5 == 5e-1 * 10
True
>>> True != False
True
>>> 5 in list(range(6))
True
>>> exit()
```

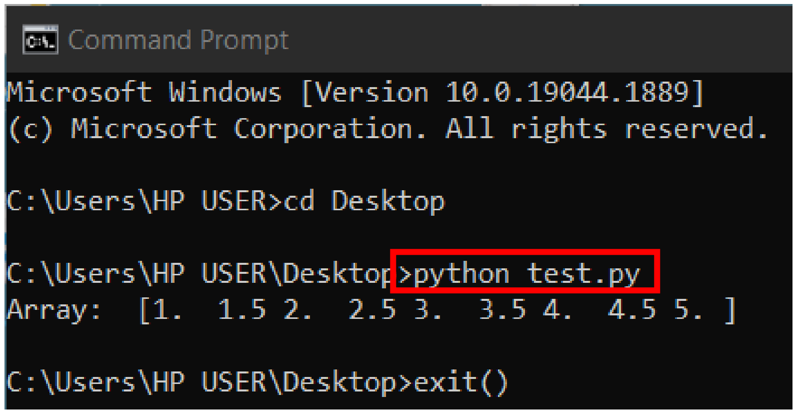
2.1 Mode of Programming

Script Mode Programming

- The executed python script begins from the top and continues to the bottom until the script is finished.
- Python files have extension **.py**



```python
import numpy as np
print("Array: ", np.linspace(1,5,9))
```



```
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP USER>cd Desktop

C:\Users\HP USER\Desktop>python test.py
Array:  [1.  1.5 2.  2.5 3.  3.5 4.  4.5 5. ]

C:\Users\HP USER\Desktop>exit()
```

2.2 Python Identifiers

- A python identifier is a name used to identify a variable, function, class module or other objects.
- A python identifier can only start with the letter A to Z or a to z or an underscore (_) followed by letters, digits (0 to 9), and underscore.
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case-sensitive programming language. Thus, **Xi** and **xi** are two different identifiers.

2.3 Reserved Words

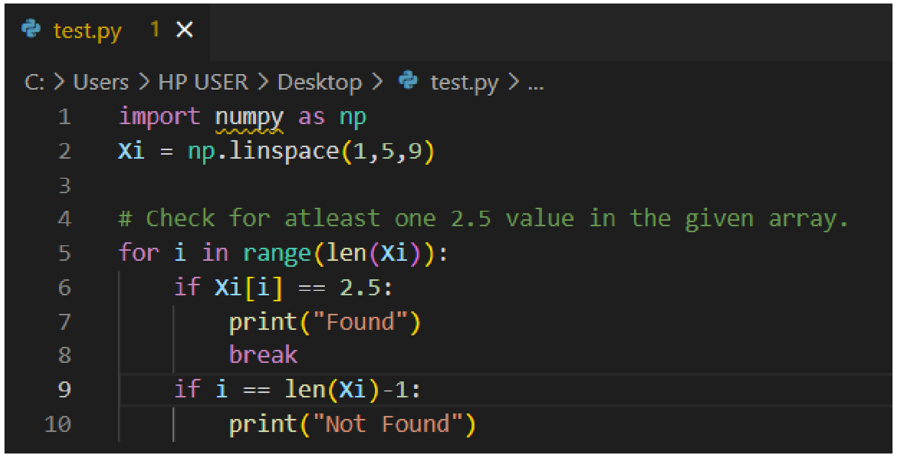
- The following list shows the python keywords. All the keywords contain lowercase letters only. These words cannot be used as constant, variable, or other identifier names.

and	else	import	print
as	except	in	raise
assert	exec	is	return
break	False	lambda	True
class	finally	none	try
continue	for	nonlocal	while
def	from	not	with
del	global	or	yield
elif	if	pass	

2.4 Lines and Indentation

Indentation

- Python has no braces to indicate blocks of code for class and function definitions. It uses line indentation to separate blocks of code.
- Tab space (8 character space) is used to separate blocks of code.

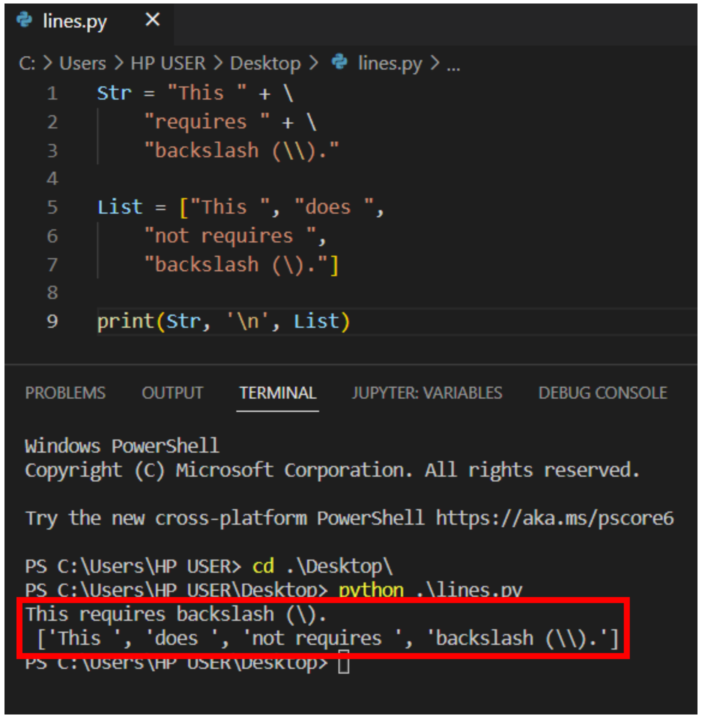


```python
import numpy as np
Xi = np.linspace(1,5,9)

# Check for atleast one 2.5 value in the given array.
for i in range(len(Xi)):
    if Xi[i] == 2.5:
        print("Found")
        break
    if i == len(Xi)-1:
        print("Not Found")
```

2.4 Lines and Indentation

Multi-line Statements

- Statements in python typically end with a new line. To use a continuous line with new lines, backslash (\) can be used.
- Statements containing [], {}, or () brackets do not require backslash.


lines.py
C: > Users > HP USER > Desktop > lines.py > ...
PROBLEMS OUTPUT TERMINAL JUPYTER: VARIABLES DEBUG CONSOLE

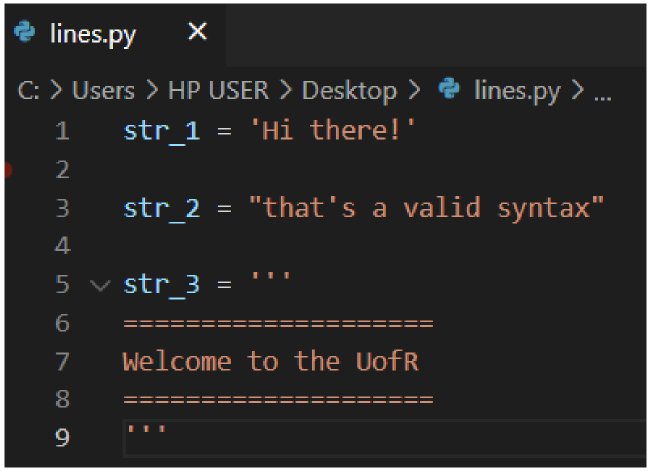
```python
Str = "This " + \
    "requires " + \
    "backslash (\\)."

List = ["This ", "does ",
    "not requires ",
    "backslash (\)."]

print(Str, '\n', List)
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\HP USER> cd .\Desktop\
PS C:\Users\HP USER\Desktop> python .\lines.py
This requires backslash (\).
 ['This ', 'does ', 'not requires ', 'backslash (\\).']
PS C:\Users\HP USER\Desktop> 
```

2.4 Lines and Indentation

Quotation in Python

- Python uses single ('), double ("), and triple quotes (''' or """) to represent string literals.
- The string syntax starts and ends with the same type of quote.
- The triple quote is used especially for multi-line strings.

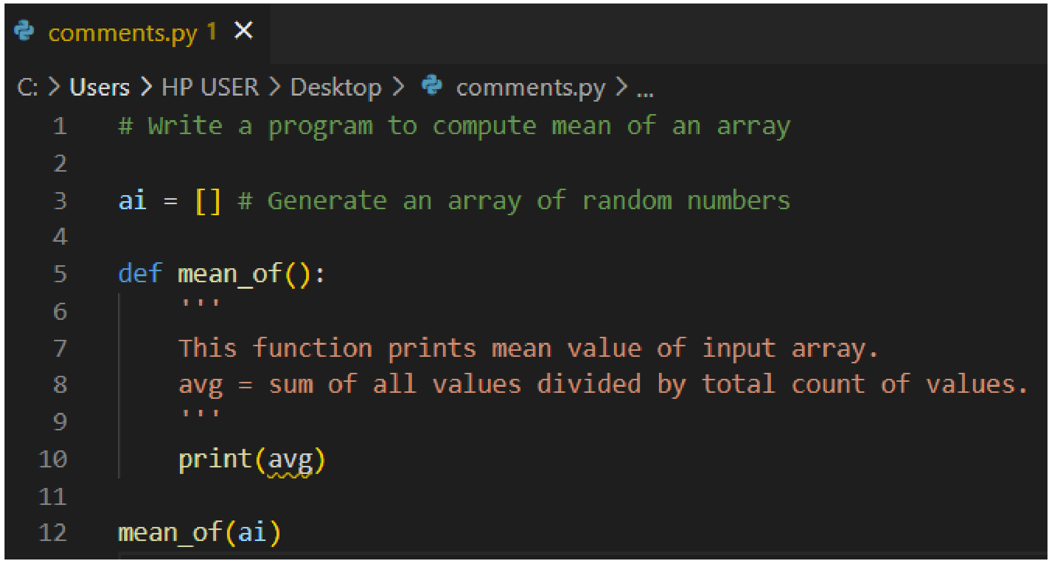


```python
str_1 = 'Hi there!'

str_2 = "that's a valid syntax"

str_3 = '''
====================
Welcome to the UofR
====================
'''
```

2.4 Lines and Indentation

Comments in Python

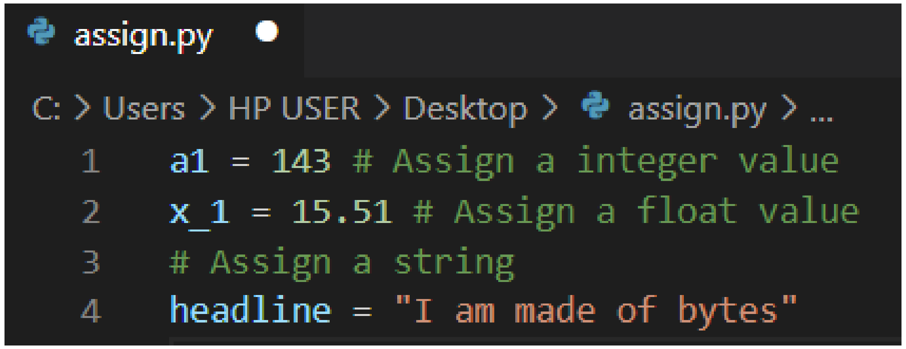
- A hash sign (#) begins a comment syntax in python. The python interpreter will ignore any statements after #.
- Triple quotes can be used for multi-line comments.



```
# Write a program to compute mean of an array

ai = [] # Generate an array of random numbers

def mean_of():
    '''
    This function prints mean value of input array.
    avg = sum of all values divided by total count of values.
    '''
    print(avg)

mean_of(ai)
```

2.5 Variable Assignment

Equal to sign (=) is used to assign values to Variables.

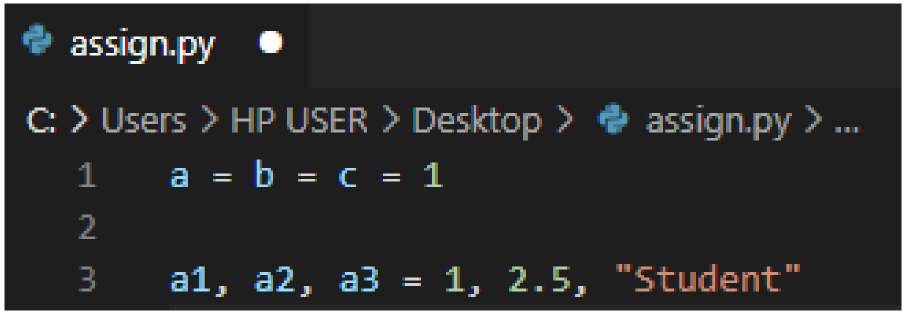


assign.py

C: > Users > HP USER > Desktop > assign.py > ...

```python
a1 = 143 # Assign a integer value
x_1 = 15.51 # Assign a float value
# Assign a string
headline = "I am made of bytes"
```

Multiple value assignment is possible as follows.



assign.py

C: > Users > HP USER > Desktop > assign.py > ...

```python
a = b = c = 1

a1, a2, a3 = 1, 2.5, "Student"
```

2.6 Variable Types

- Python has a wide range of built-in data types.
- Function `type()` can be used to determine the data type.
- The most commonly used data types are Numeric, Text, Sequence, and Dictionary data categories.

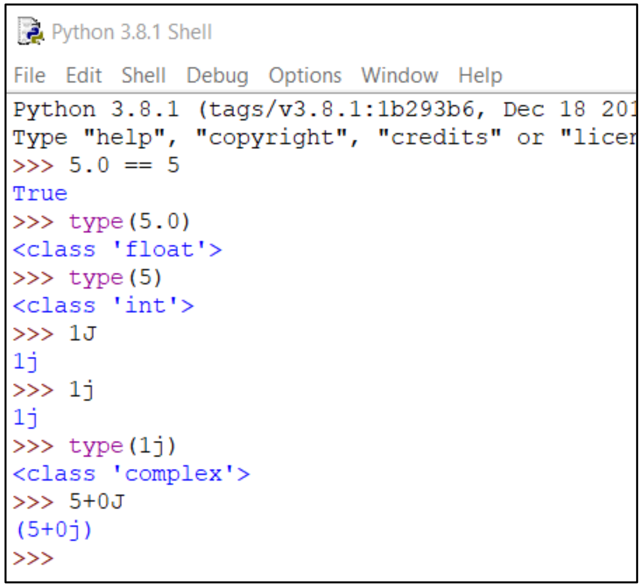
Python Number:	`int, float, complex`
Python String:	`str`
Python Sequence:	`list, tuple`
Python Dictionary:	`dict`

2.6 Variable Types

Python Number

- Three types of numerical data types are accepted.
- **Integer** is a whole number with positive and negative numbers, without decimal values.
- **Float** is an integer with decimal values.
- **Complex** is a complex number with "j" as the imaginary part.
- Examples of `int`, `float`, `complex` data types are as follows.

int	float	complex
10	0.0	3.14j
100	14.80	-.625+0j
-789	-21.9	3e+26J
80	3.6e+5	1+1j
-253	-50.	0.5*(1+1j)



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 201
Type "help", "copyright", "credits" or "licer
>>> 5.0 == 5
True
>>> type(5.0)
<class 'float'>
>>> type(5)
<class 'int'>
>>> 1J
1j
>>> 1j
1j
>>> type(1j)
<class 'complex'>
>>> 5+0J
(5+0j)
>>>
```

2.6 Variable Types

Python Number built-in methods

Method	Description
abs()	Returns absolute value
complex()	Converts data type into an complex number
float()	Converts data type into a floating point number
int()	Converts data type into an integer number
pow(x, y)	Returns x to the power of y value. Similar to x** y operation
pow(x, y, z)	Returns (x ** y) % z value
round(x, y)	Converts x to y decimal count(s)
.is_integer()	Returns True if the value is an integer
.real	Returns real component of the imaginary number
.imag	Returns imaginary component of the imaginary number
.conjugate()	Converts the sign of the imaginary component of the imaginary number

2.6 Variable Types

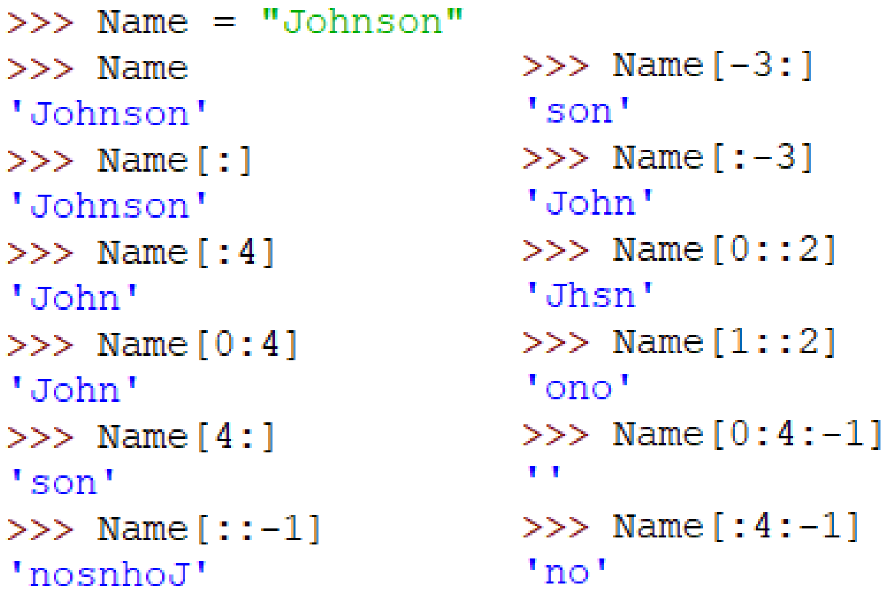
Python String

- Strings in Python are identified as a set of characters represented in quotation marks.
- `Str` data type is used to represent a string in python.
- Syntax of python string is previously explained here.
- Subsets of strings can be obtained using the slice operator ([:]).
- The syntax of the slice operator is as follows [start:end:step]
- Start and end are index values, whereas step is a consecutive selection of characters.

2.6 Variable Types

Python String slicing operator

Negative Index	-7	-6	-5	-4	-3	-2	-1
String	J	o	h	n	s	o	n
Index	0	1	2	3	4	5	6



```
>>> Name = "Johnson"
>>> Name
'Johnson'
>>> Name[:]
'Johnson'
>>> Name[:4]
'John'
>>> Name[0:4]
'John'
>>> Name[4:]
'son'
>>> Name[::-1]
'nosnhoJ'
```

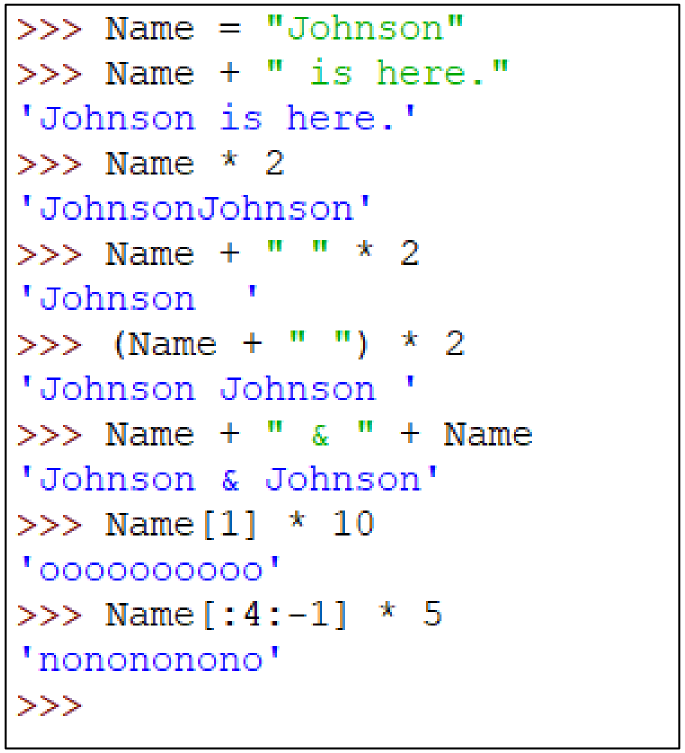
```
>>> Name[-3:]
'son'
>>> Name[:-3]
'John'
>>> Name[0::2]
'Jhsn'
>>> Name[1::2]
'ono'
>>> Name[0:4:-1]
''
>>> Name[:4:-1]
'no'
```

2.6 Variable Types

Python String other operators

- The plus sign (+) for string concatenation.
- The asterisk sign (*) for string repetition.

- Here are some examples.



```
>>> Name = "Johnson"
>>> Name + " is here."
'Johnson is here.'
>>> Name * 2
'JohnsonJohnson'
>>> Name + " " * 2
'Johnson  '
>>>  (Name + " ") * 2
'Johnson Johnson '
>>> Name + " & " + Name
'Johnson & Johnson'
>>> Name[1] * 10
'oooooooooo'
>>> Name[:4:-1] * 5
'nonononono'
>>>
```

2.6 Variable Types

Python String built-in methods

Method	Description
len()	Returns count of characters in the string
str()	Converts data type into string
.capitalize()	Converts the first character to upper case
.casefold()	Converts string into lower case
.center()	Returns a centered string
.count()	Returns the number of times a specified value occurs in a string
.encode()	Returns an encoded version of the string
.endswith()	Returns true if the string ends with the specified value
.expandtabs()	Sets the tab size of the string
.find()	Searches the string for a specified value and returns the position of where it was found

2.6 Variable Types

Python String built-in methods

Method	Description
.format()	Formats specified values in a string
.format_map()	Formats specified values in a string
.index()	Searches the string for a specified value and returns the position of where it was found
.isalnum()	Returns True if all characters in the string are alphanumeric
.isalpha()	Returns True if all characters in the string are in the alphabet
.isdecimal()	Returns True if all characters in the string are decimals
.isdigit()	Returns True if all characters in the string are digits
.isidentifier()	Returns True if the string is an identifier
.islower()	Returns True if all characters in the string are lower case
.isnumeric()	Returns True if all characters in the string are numeric

2.6 Variable Types

Python String built-in methods

Method	Description
.isprintable()	Returns True if all characters in the string are printable
.isspace()	Returns True if all characters in the string are whitespaces
.istitle()	Returns True if the string follows the rules of a title
.isupper()	Returns True if all characters in the string are upper case
.join()	Joins the elements of an iterable to the end of the string
.ljust()	Returns a left justified version of the string
.lower()	Converts a string into lower case
.lstrip()	Returns a left trim version of the string
.maketrans()	Returns a translation table to be used in translations
.partition()	Returns a tuple where the string is parted into three parts

2.6 Variable Types

Python String built-in methods

Method	Description
.replace()	Returns a string where a specified value is replaced with a specified value
.rfind()	Searches the string for a specified value and returns the last position of where it was found
.rindex()	Searches the string for a specified value and returns the last position of where it was found
.rjust()	Returns a right justified version of the string
.rpartition()	Returns a tuple where the string is parted into three parts
.rsplit()	Splits the string at the specified separator, and returns a list
.rstrip()	Returns a right trim version of the string
.split()	Splits the string at the specified separator, and returns a list
.splitlines()	Splits the string at line breaks and returns a list
.startswith()	Returns true if the string starts with the specified value

2.6 Variable Types

Python String built-in methods

Method	Description
.strip()	Returns a trimmed version of the string
.swapcase()	Swaps cases, lower case becomes upper case and vice versa
.title()	Converts the first character of each word to upper case
.translate()	Returns a translated string
.upper()	Converts a string into upper case
.zfill()	Fills the string with a specified number of 0 values at the beginning

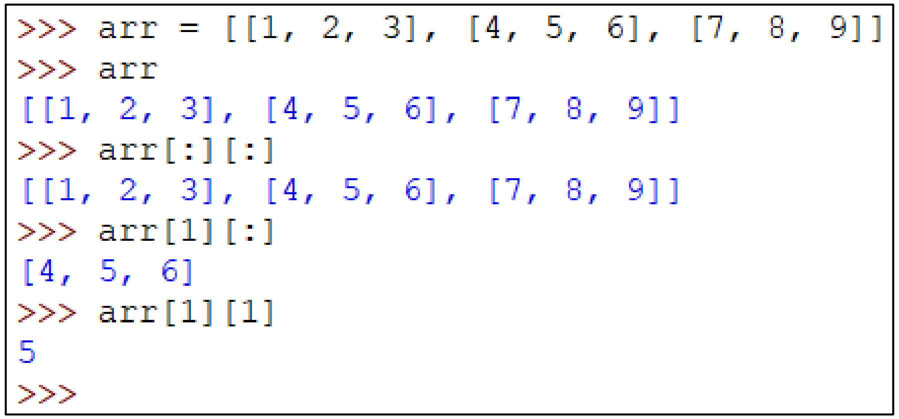
2.6 Variable Types

Python List

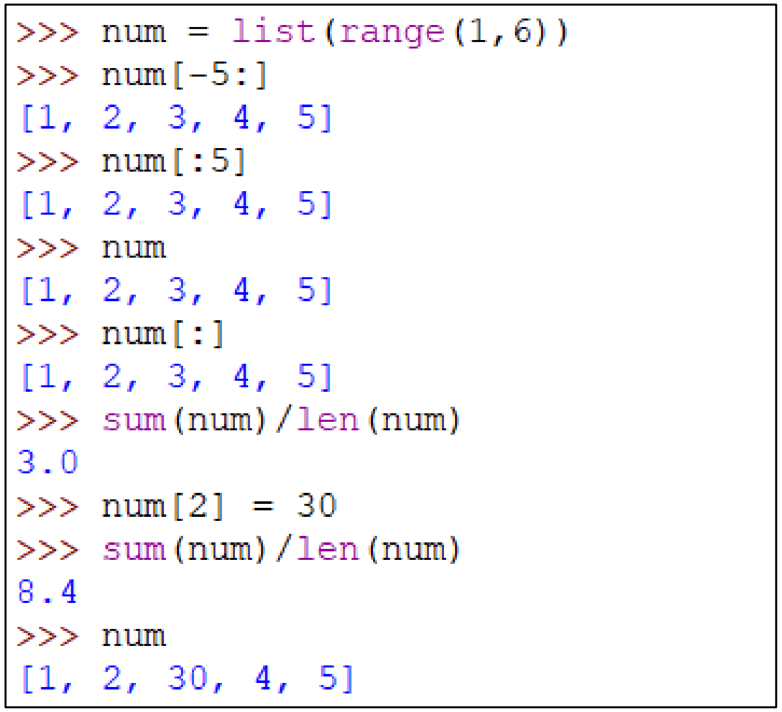
- List is a most versatile datatype, which can be created as a list of comma-separated values (items) between square brackets ([])
- `list` data type is used to represent list in python
- List accepts items of any data type
- List items are ordered, changeable, and allow duplicate values

2.6 Variable Types

Examples of python List



```
>>> arr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> arr
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> arr[:][:]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> arr[1][:]
[4, 5, 6]
>>> arr[1][1]
5
>>>
```



```
>>> num = list(range(1,6))
>>> num[-5:]
[1, 2, 3, 4, 5]
>>> num[:5]
[1, 2, 3, 4, 5]
>>> num
[1, 2, 3, 4, 5]
>>> num[:]
[1, 2, 3, 4, 5]
>>> sum(num)/len(num)
3.0
>>> num[2] = 30
>>> sum(num)/len(num)
8.4
>>> num
[1, 2, 30, 4, 5]
```

2.6 Variable Types

Python List built-in methods

Method	Description
cmp(list1, list2)	Compares elements of both lists
len()	Returns count of elements in list
list()	Converts data type into list
max()	Returns item from the list with maximum value
min()	Returns item from the list with minimum value
.append()	Adds an element at the end of the list
.clear()	Removes all the elements from the list
.copy()	Returns a copy of the list
.count()	Returns the number of elements with the specified value
.extend()	Add the elements of a list (or any iterable), to the end of the current list

2.6 Variable Types

Python List built-in methods

Method	Description
.index()	Returns the index of the first element with the specified value
.insert()	Adds an element at the specified position
.pop()	Removes the element at the specified position
.remove()	Removes the item with the specified value
.reverse()	Reverses the order of the list
.sort()	Sorts the list

2.6 Variable Types

Python Tuple

- Tuple can be created as a list of comma-separated values (items) between round brackets ()
- `tuple` data type is used to represent tuple in python
- Tuple accepts items of any data type
- Tuple items are ordered and unchangeable

2.6 Variable Types

Python Tuple build-in methods

Method	Description
tuple()	Converts data type to tuple
.count()	Returns the number of times a specified value occurs in a tuple
.index()	Searches the tuple for a specified value and returns the position of where it was found

2.6 Variable Types

Python Dictionary

- Dictionary can be created as a pair of key: value between round brackets ({})
- `dict` data type is used to represent dictionary in python
- Dictionary accepts items of any data type
- Dictionary items are ordered, changeable, and do not allow duplicates
- examples

2.6 Variable Types

Python Dictionary built-in methods

Method	Description
dict()	Converts data type into dictionary
.clear()	Removes all the elements from the dictionary
.copy()	Returns a copy of the dictionary
.fromkeys()	Returns a dictionary with the specified keys and value
.get()	Returns the value of the specified key
.items()	Returns a list containing a tuple for each key value pair
.keys()	Returns a list containing the dictionary's keys
.pop()	Removes the element with the specified key
.popitem()	Removes the last inserted key-value pair

2.6 Variable Types

Python Dictionary built-in methods

Method	Description
.setdefault()	Returns the value of the specified key. If the key does not exist, insert the key with the specified value
.update()	Updates the dictionary with the specified key-value pairs
.values()	Returns a list of all the values in the dictionary

2.7 List of built-in Methods

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	vars()
chr()	frozenset()	list()	range()	zip()
classmethod()	getattr()	locals()	repr()	
compile()	globals()	map()	reversed()	
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

2.8 Basic Operators

Operators are the constructs which can manipulate the value of operands.

Types of operators:

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Assignment Operators
4. Logical Operators
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

2.8.1 Arithmetic Operators

Operator	Description	Example
+	**Addition:** Adds values on either side of the operator	a = 10, b = 20 a + b = 30
-	**Subtraction:** Subtracts values on either side of the operator	a - b = -10
*	**Multiplication:** Multiplies values on either side of the operator	a*b = 200
/	**Division:** Divides left hand operand by right hand operand	b/a = 2
%	**Modulus:** Divides left hand operand by right hand operand and returns reminder	b%a = 0
**	**Exponent:** Performs exponential (power) calculation on operators	a**b = 10^20
//	**Floor Division:** Return absolute value of division operator	b//a = 2

2.8.2 Comparison (Relational) Operators

Operator	Description	Example
==	**Equal to:** If values of two operands are equal, then the condition becomes true	a = 10, b = 20 a == b False
!=	**Not equal to:** If values of two operands are not equal, then the condition becomes true	a != b True
>	**Greater than:** If the value of left operand is greater than the value of right operand, then the condition becomes true	a > b False
<	**Less than:** If the value of left operand is less than the value of right operand, then the condition becomes true	a < b True
>=	**Greater than or equal to:** If the value of left operand is greater than or equal to the value of right operand, then the condition becomes true	a >= b False
<=	**Less than or equal to:** If the value of left operand is less than or equal to the value of right operand, then the condition becomes true	a <= b True

2.8.3 Assignment Operators

Operator	Description	Example
=	**Assign** value from right side operands to left side operand	c = a + b assigns value if a + b into c
+=	**Add AND:** adds right operand to the left operand and assigns the result to left operand	c += a is equivalent to c = c + a
-=	**Subtract AND:** subtracts right operand to the left operand and assigns the result to left operand	c -= a is equivalent to c = c - a
*=	**Multiply AND:** multiplies right operand to the left operand and assigns the result to left operand	c *= a is equivalent to c = c * a
/=	**Divide AND:** divides right operand to the left operand and assigns the result to left operand	c /= a is equivalent to c = c / a
%=	**Modulus AND:** performs reminder of divide operator and assigns the result to left operand	c %= a is equivalent to c = c % a
**=	**Exponent AND:** performs exponential (power) calculations on two operands and assigns the result to left operand	c **= a is equivalent to c = c ** a
//=	**Floor Division AND:** performs floor division on two operands and assigns the result to left operand	c //= a is equivalent to c = c // a

2.8.4 Logical Operators

Operator	Description	Example
and	**Logical AND:** if both the operands are true then condition becomes true	a and b True
or	**Logical OR:** if any of the two operands are non-zero then condition becomes true	a or b False
not	**Logical NOT:** Used to reverse the logical state of its operand	not(True) False

2.8.5 Bitwise Operators

Operator	Description	Example

&	**Binary AND:** Performs AND logical operation bit by bit	a = 0011 1100
		b = 0000 1101
		a&b = 0000 1100

|	**Binary OR:** Performs OR logical operation bit by bit	a|b = 0011 1101
^	**Binary XOR:** Performs XOR logical operation bit by bit	a^b = 0011 0001
~	**Binary Ones Complement:** Performs NOT logical operation bit by bit	~a = 1100 0011
<<	**Binary Left Shift:** values are moved left by the number of bits specified by the right operand	a<<2 = 1111 0000
>>	**Binary Right Shift:** values are moved right by the number of bits specified by the right operand	a>>2 = 0000 1111

2.8.6 Membership Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise	3 in [1, 3, 5] True
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise	4 not in [1, 3, 5] True

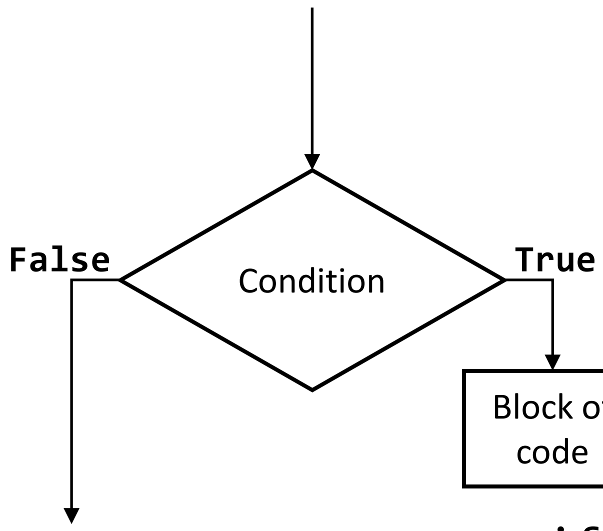
2.8.7 Identity Operators

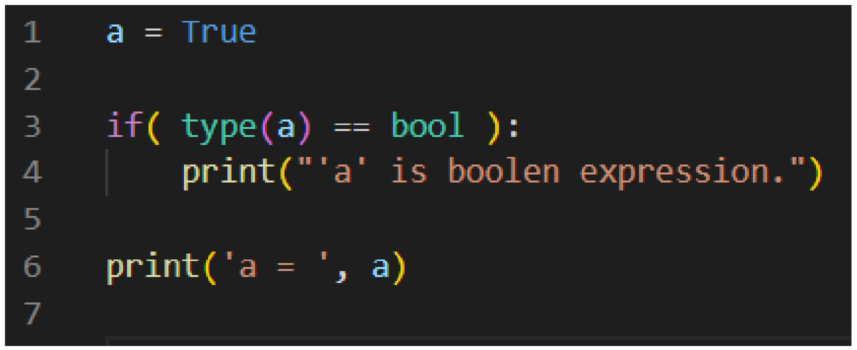
Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise	[1, 3, 5] is [1, 3, 5] True
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise	[0, 3, 5] is not [1, 3, 5] True

2.9 Conditional Operators

if statement

- Decision making in python is performed by if conditional operator.

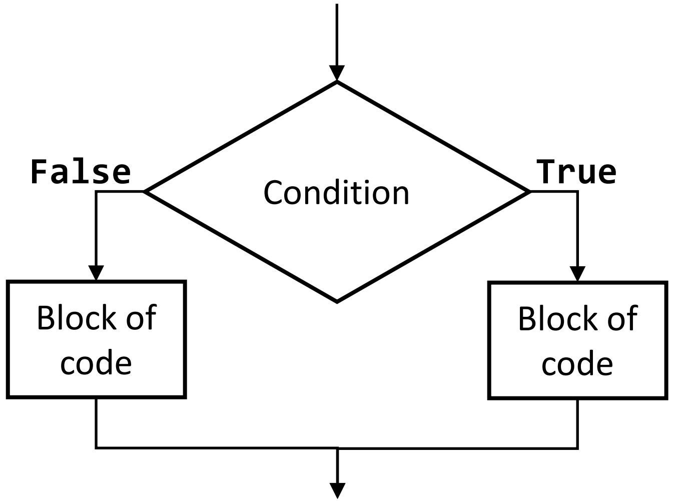


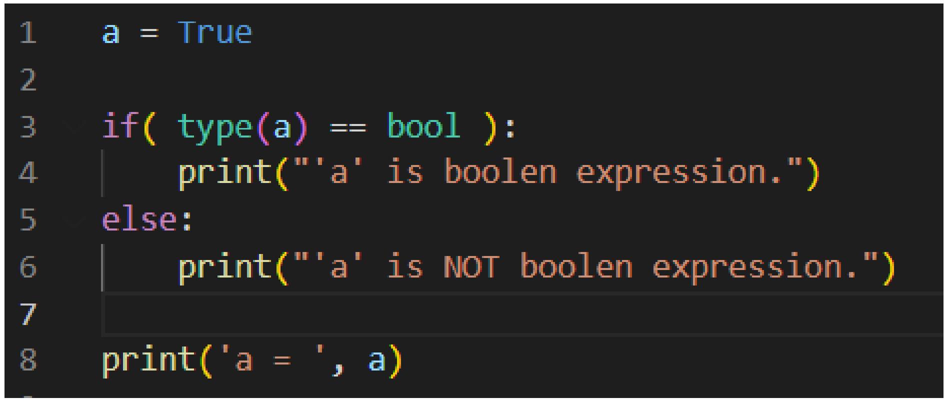


```
a = True

if( type(a) == bool ):
    print("'a' is boolen expression.")

print('a = ', a)

```

`if` condition example

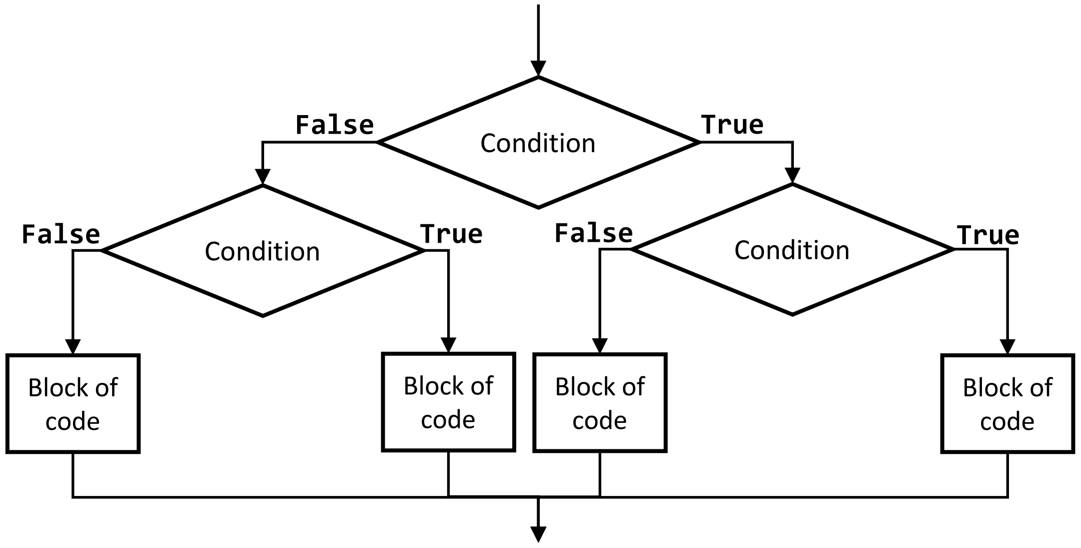
2.9 Conditional Operators

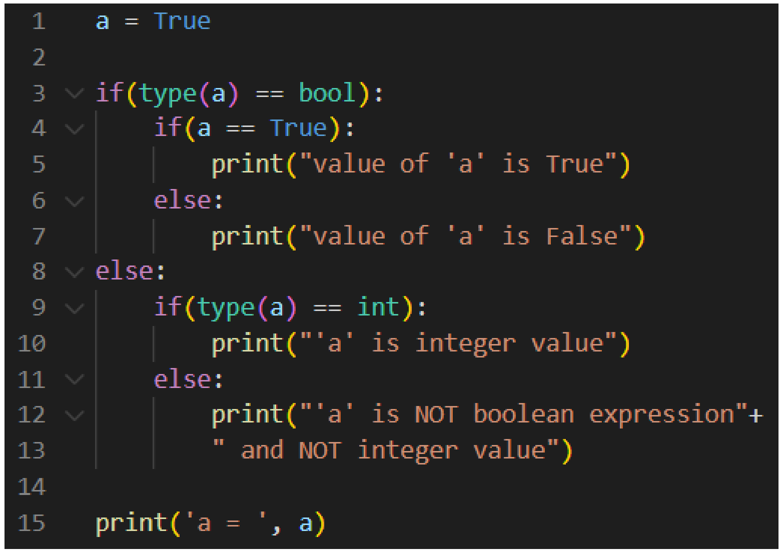




```python
a = True

if( type(a) == bool ):
    print("'a' is boolen expression.")
else:
    print("'a' is NOT boolen expression.")

print('a = ', a)
```

`if else` condition example

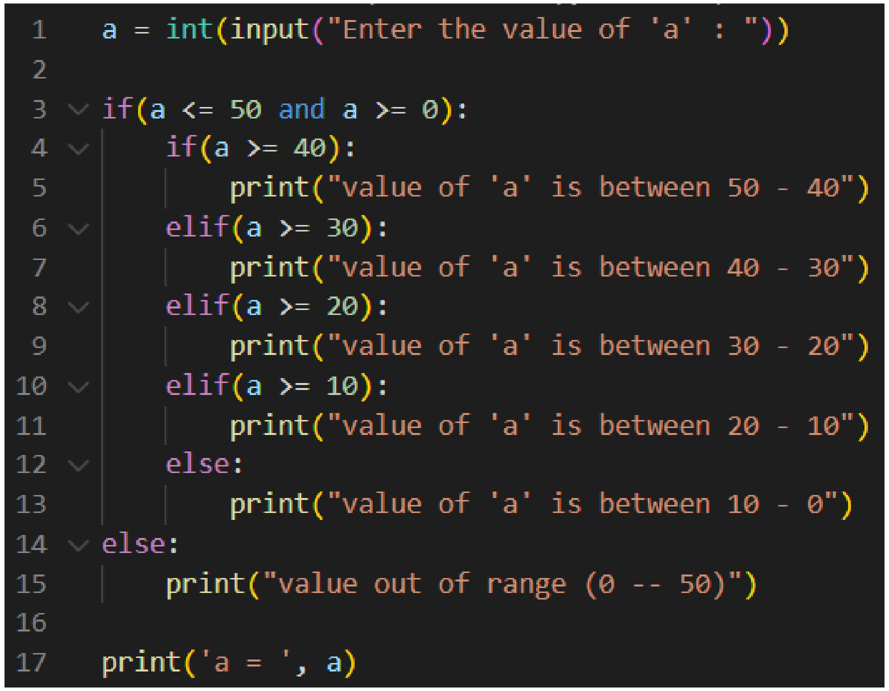




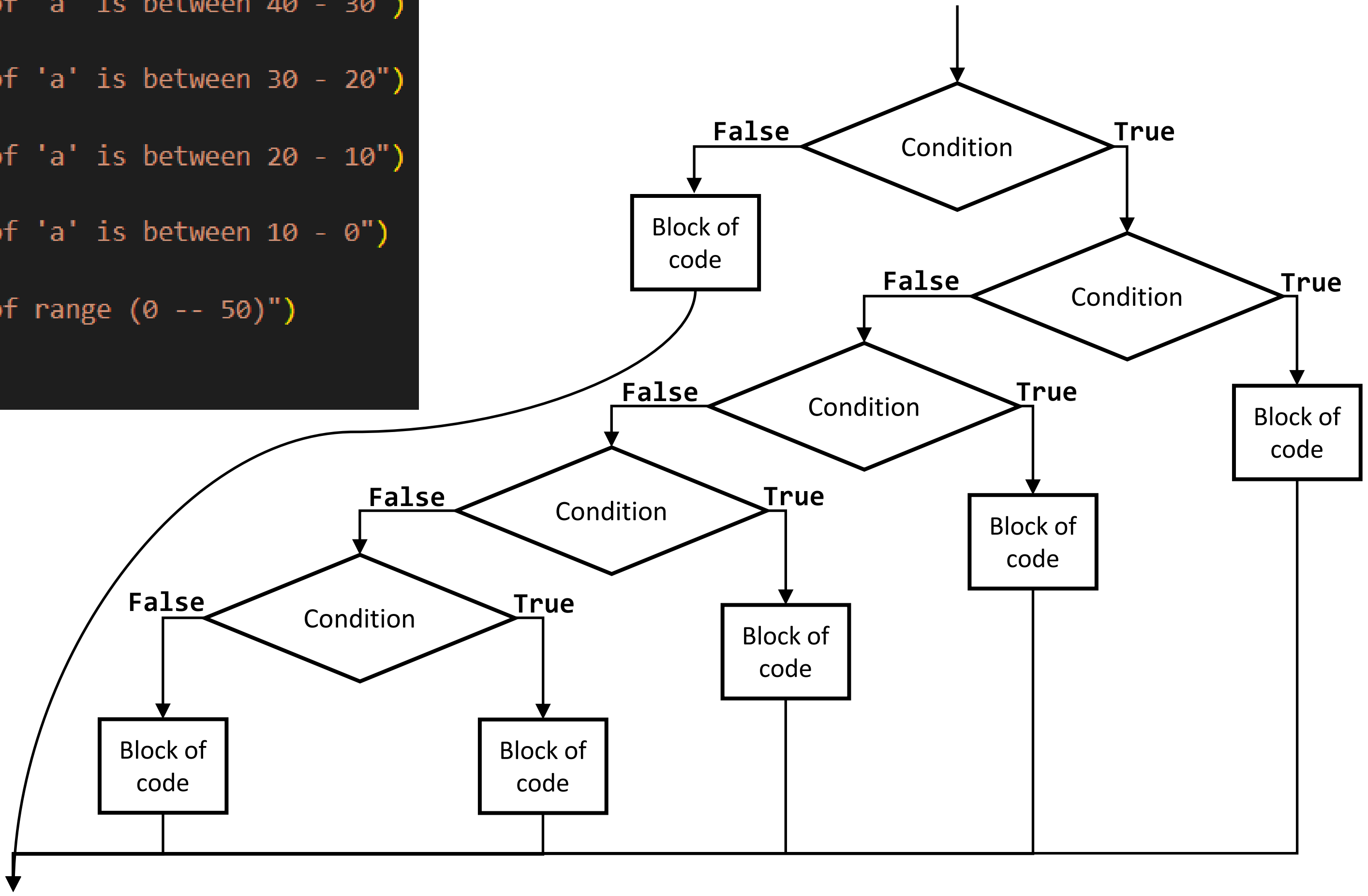
```python
a = True

if(type(a) == bool):
    if(a == True):
        print("value of 'a' is True")
    else:
        print("value of 'a' is False")
else:
    if(type(a) == int):
        print("'a' is integer value")
    else:
        print("'a' is NOT boolean expression"+
        " and NOT integer value")

print('a = ', a)
```

Nested `if` condition example 01

2.9 Conditional Operators



```python
a = int(input("Enter the value of 'a' : "))

if(a <= 50 and a >= 0):
    if(a >= 40):
        print("value of 'a' is between 50 - 40")
    elif(a >= 30):
        print("value of 'a' is between 40 - 30")
    elif(a >= 20):
        print("value of 'a' is between 30 - 20")
    elif(a >= 10):
        print("value of 'a' is between 20 - 10")
    else:
        print("value of 'a' is between 10 - 0")
else:
    print("value out of range (0 -- 50)")

print('a = ', a)
```

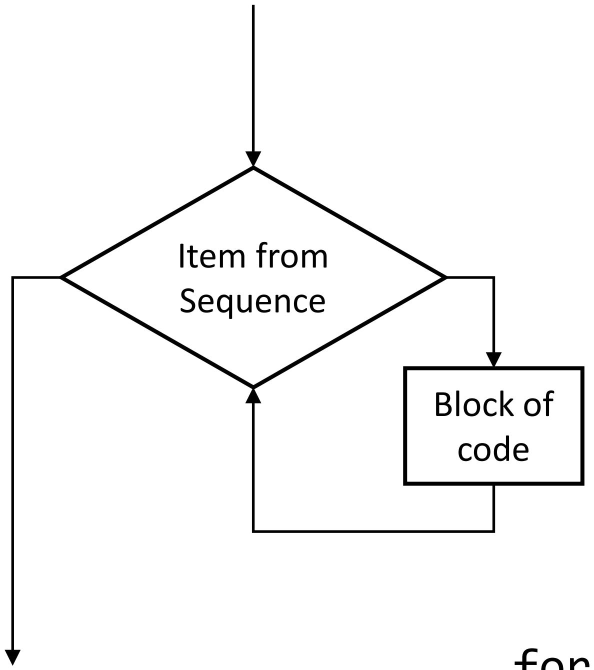


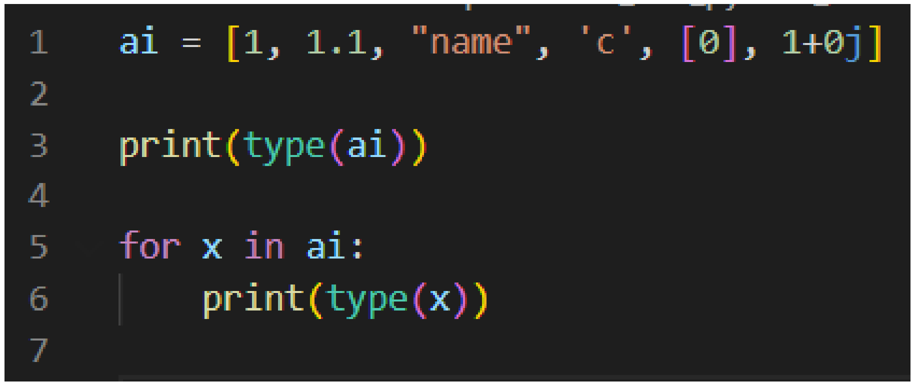
Nested `if` condition example 02

2.10 Loop Statements

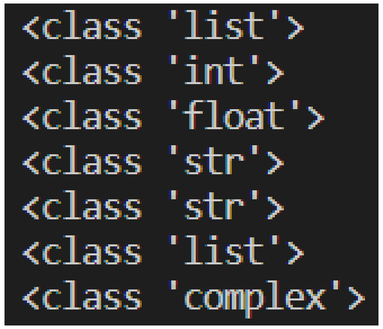
for loop

- for statement has the ability to iterate over the items of any sequence, such as a list or a string.





```python
ai = [1, 1.1, "name", 'c', [0], 1+0j]

print(type(ai))

for x in ai:
    print(type(x))

```



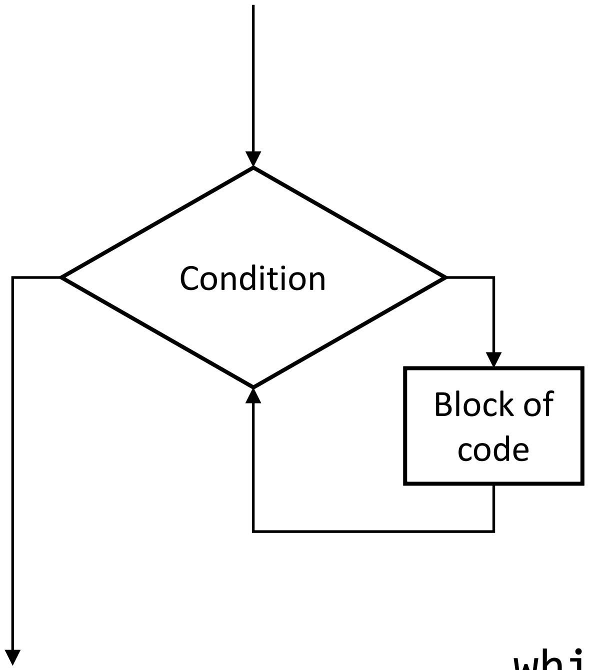
```
<class 'list'>
<class 'int'>
<class 'float'>
<class 'str'>
<class 'str'>
<class 'list'>
<class 'complex'>
```

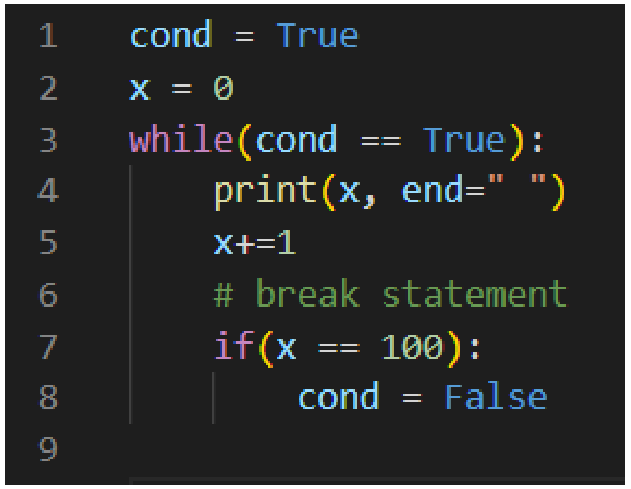
for loop example

2.10 Loop Statements

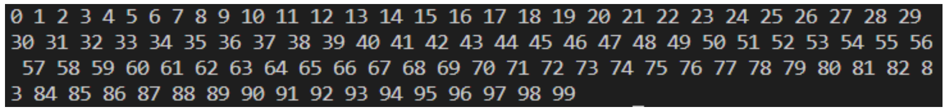
while loop

while statement repeatedly executes a target statement as long as a given condition is true





```python
cond = True
x = 0
while(cond == True):
    print(x, end=" ")
    x+=1
    # break statement
    if(x == 100):
        cond = False

```



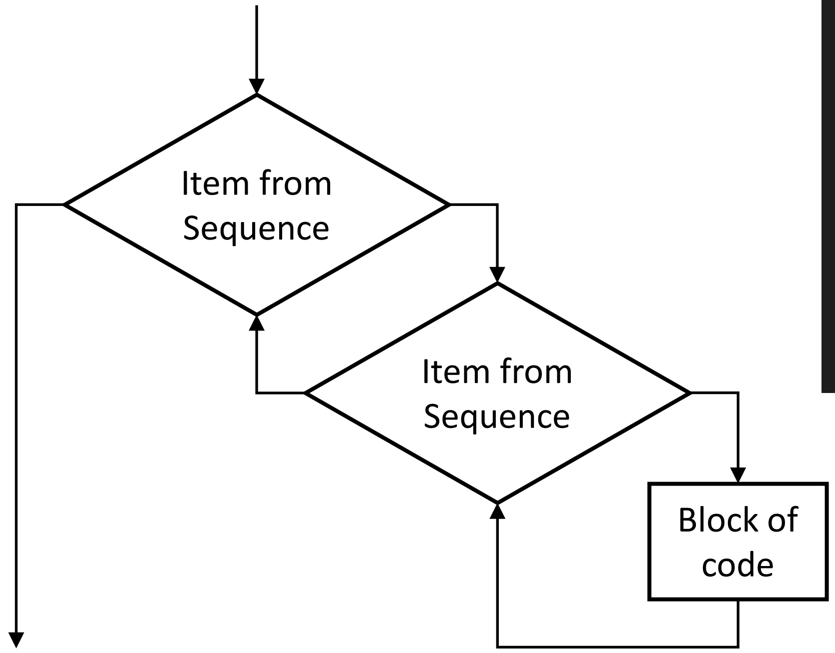
```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 8
3 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
```

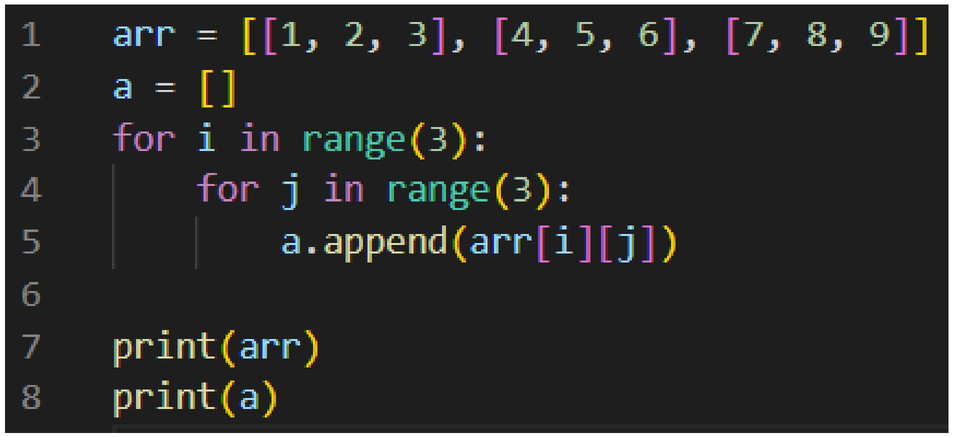
while loop example

2.10 Loop Statements

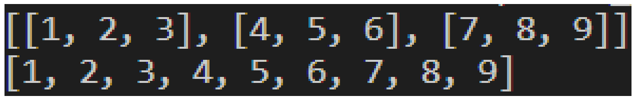
nested loops

- Intending a loop inside another loop is allowed in python





```
arr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
a = []
for i in range(3):
    for j in range(3):
        a.append(arr[i][j])

print(arr)
print(a)
```



```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Nested `for` loop example

2.10 Loop Statements

Loop Control Statements

Statements	Description
break	Terminates the loop statement and transfers execution to the statement immediately following the loop
continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating
pass	Used when a statement is required syntactically but you do not want any command or code to execute

2.11 Files I/O

- The `open(file, mode)` function performs file handling in python.
- In the syntax, the `file` specifies the directory and file name. The `mode` has the following string options on the file accessed.

Mode	Description
"r"	Read: Default value. Opens a file for reading.
"a"	Append: Opens a file for appending and creates the file if it does not exist.
"w"	Write: Opens a file for writing, creates the file if it does not exist.
"x"	Create: Create the specified file and return s an error if the file exists.
"t"	Text Mode: In addition to the above options, the file data type can be specified.
"b"	Binary Mode: In addition to the above options, the file data type can be specified.
"+"	Additional Mode: In addition to the above options, both the first specified mode and reading modes are enabled.

2.11 Files I/O

open() built-in methods

Method	Description
.closed	Returns true if file is closed, false otherwise
.mode	Returns access mode with which file was opened
.name	Returns name of the file
.softspace	Returns false if space explicitly required with print, true otherwise
.close()	Close the file
.write()	Write string into the file
.read()	Read the content of the file
.tell()	Returns current position within the file
.seek()	Change the position within the file

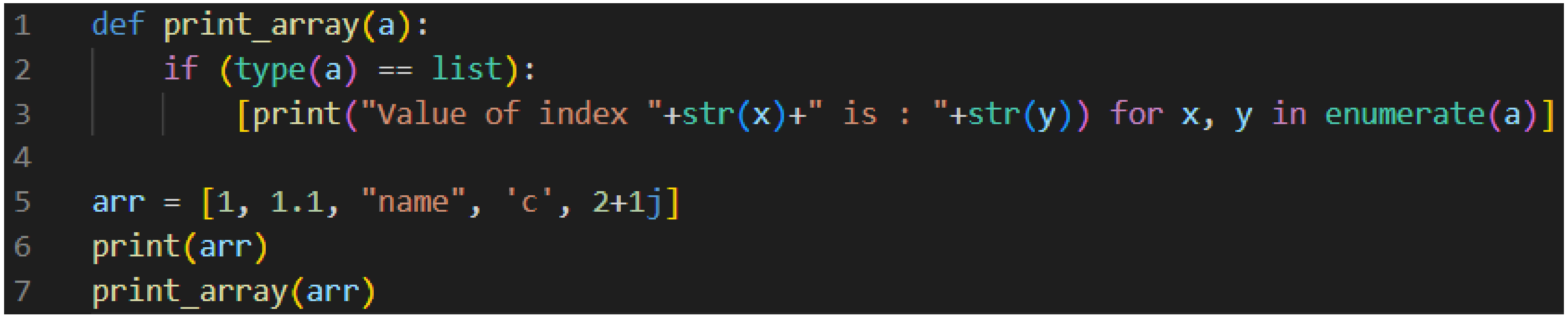
2.12 Functions

Defining a function

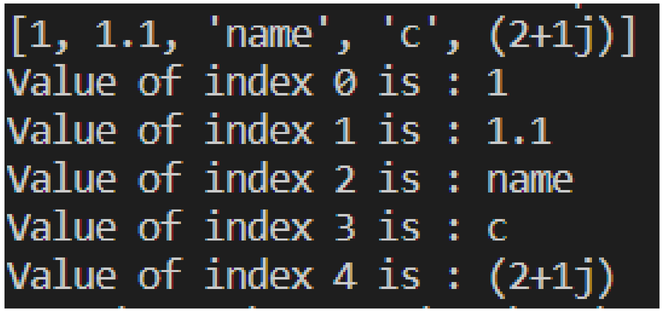
- A function is a block of organized, reusable code to perform a single, related action.
- A user-defined function can be created in python by using the keyword def followed by the function name, parentheses (()) and colon (:).
- The first statement of a function can be a documentation string called a docstring.
- The **return** statement exits a function. Optionally the function can be exited by indentation.

2.12 Functions

Example of python function



```python
def print_array(a):
    if (type(a) == list):
        [print("Value of index "+str(x)+" is : "+str(y)) for x, y in enumerate(a)]

arr = [1, 1.1, "name", 'c', 2+1j]
print(arr)
print_array(arr)
```

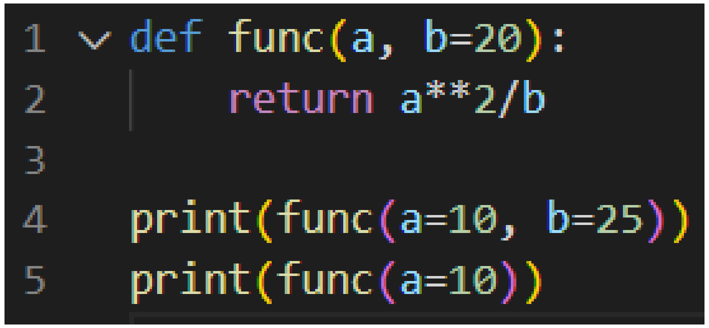


```
[1, 1.1, 'name', 'c', (2+1j)]
Value of index 0 is : 1
Value of index 1 is : 1.1
Value of index 2 is : name
Value of index 3 is : c
Value of index 4 is : (2+1j)
```

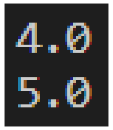
function example 01

2.12 Functions

Example of python function

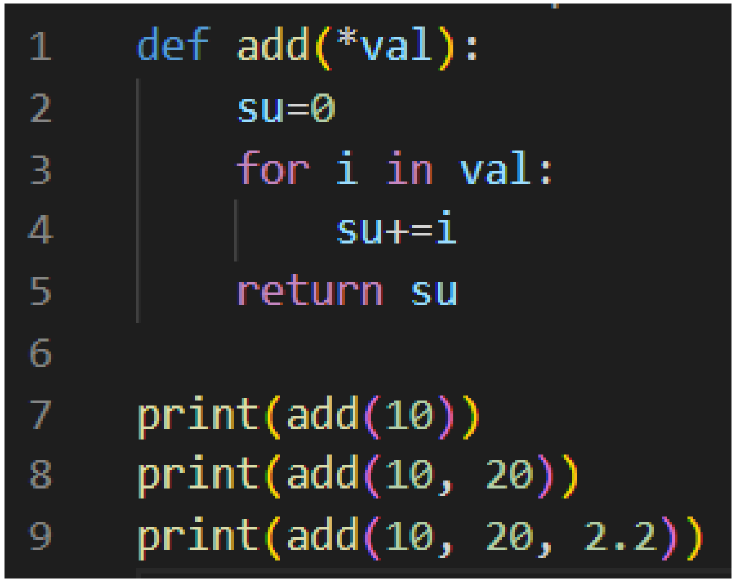


```python
def func(a, b=20):
    return a**2/b

print(func(a=10, b=25))
print(func(a=10))
```

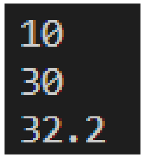


```
4.0
5.0
```

function example 02



```python
def add(*val):
    su=0
    for i in val:
        su+=i
    return su

print(add(10))
print(add(10, 20))
print(add(10, 20, 2.2))
```



```
10
30
32.2
```

function example 03

2.12 Functions

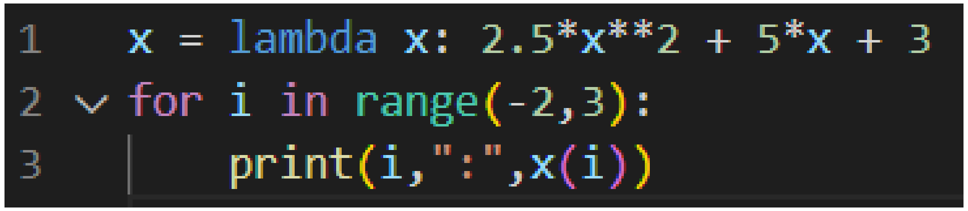
Function Arguments

Arguments	Description	Example
Required Arguments	Arguments passed to a function in correct positional order	print_array()
Keyword Arguments	Arguments passed to a function by the parameter name	func(a=10, b=25)
Default Arguments	An optional arguments which assigns default values incase of argument not passed	func(a=10)
Variable-length Arguments	An optional arguments with defined code when assigned. It has arguments beginning with *args or **kwargs	add()

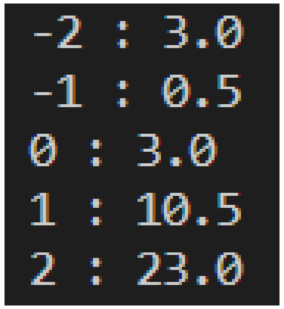
2.12 Functions

lambda (Anonymous) Function

- Lambda functions are called anonymous as they are not declared in the standard manner by using the def keyword.



```python
x = lambda x: 2.5*x**2 + 5*x + 3
for i in range(-2,3):
    print(i,":",x(i))
```



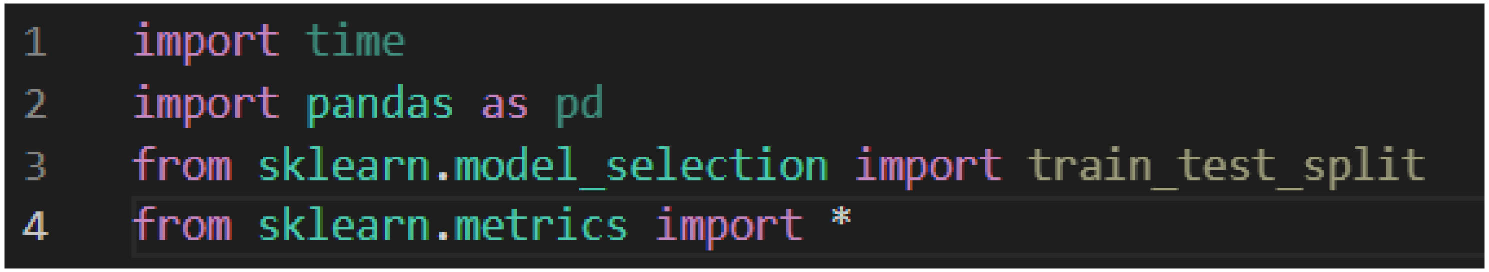
```
-2 : 3.0
-1 : 0.5
0 : 3.0
1 : 10.5
2 : 23.0
```

function example 04

2.13 Python Modules

A module allows you to logically organize your python code. Grouping related code into a module males the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Importing modules



```python
import time
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
```

2.13 Python Modules

```python
import numpy as np

A = np.random.randint(10, size=[3,3])
I = np.identity(3)
A
```

```
array([[4, 7, 8],
       [9, 5, 2],
       [2, 2, 8]])
```

```python
# determinent of A
np.linalg.det(A)
```

```
-268.00000000000017
```

```python
# matrix multiplication
np.dot(A,A)
```

```
array([[ 95,  79, 110],
       [ 85,  92,  98],
       [ 42,  40,  84]])
```

```python
np.dot(A,I)
```

```
array([[4., 7., 8.],
       [9., 5., 2.],
       [2., 2., 8.]])
```

Example of modules usage

```python
# inverse matrix of A
np.linalg.inv(A)
```

```
array([[-0.13432836,  0.14925373,  0.09701493],
       [ 0.25373134, -0.05970149, -0.23880597],
       [-0.02985075, -0.02238806,  0.16044776]])
```

```python
# eigen values and eigen vectors of A
np.linalg.eig(A)
```

```
(array([-3.40987786, 15.25920326,  5.15067459]),
 array([[ 0.68168192, -0.66421808, -0.15439474],
        [-0.73159635, -0.65346586, -0.75445171],
        [ 0.00874934, -0.36303817,  0.63793799]]))
```

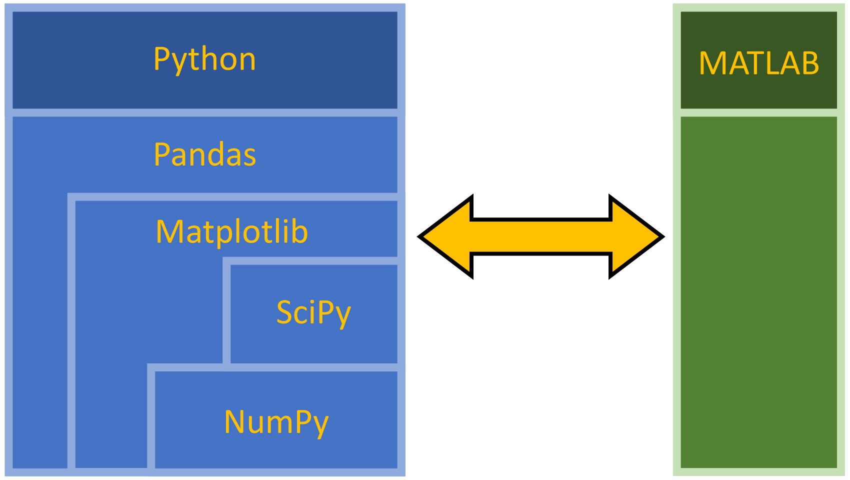
2.13 Python Modules

Packages in Python

- A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-sub packages.
- Search for Python Packages
  - PyPi: Python Package Index.
  - Available online at: https://pypi.org/

3. Essential Python packages for Numerical Methods

Python combined with Numpy, Scipy, Matplotlib, and Pandas can be used as a complete replacement for MATLAB.



Python vs MATLAB

Kong, Q., Siauw, T., & Bayen, A. (2020). *Python Programming and Numerical Methods: A Guide for Engineers and Scientists*. Academic Press. Available online at: https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html

3. Essential Python packages for Numerical Methods

1. NumPy is a module which provides the basic data structures, implementing multi-dimensional arrays and matrices. Besides that, the modules supply the necessary functionalities to create and manipulate these data structures.
2. Matplotlib is a plotting library for the Python programming language and numerically oriented modules like NumPy and SciPy. The youngest child in this family of modules is Pandas.
3. Pandas is using all of the previously mentioned modules. It’s built on top of them to provide a module for the Python language, which is also capable of data manipulation and analysis. The special focus of Pandas consists in offering data structures and operations for manipulating numerical tables and time series. The name is derived from the term "panel data.” Pandas is well suited for working with tabular data as it is known from spreadsheet programming like Excel.
4. SciPy is based on top of NumPy; it uses the data structures provided by NumPy. It extends the capabilities of NumPy with additional useful functions for minimization, regression, Fourier-transformation and many others.