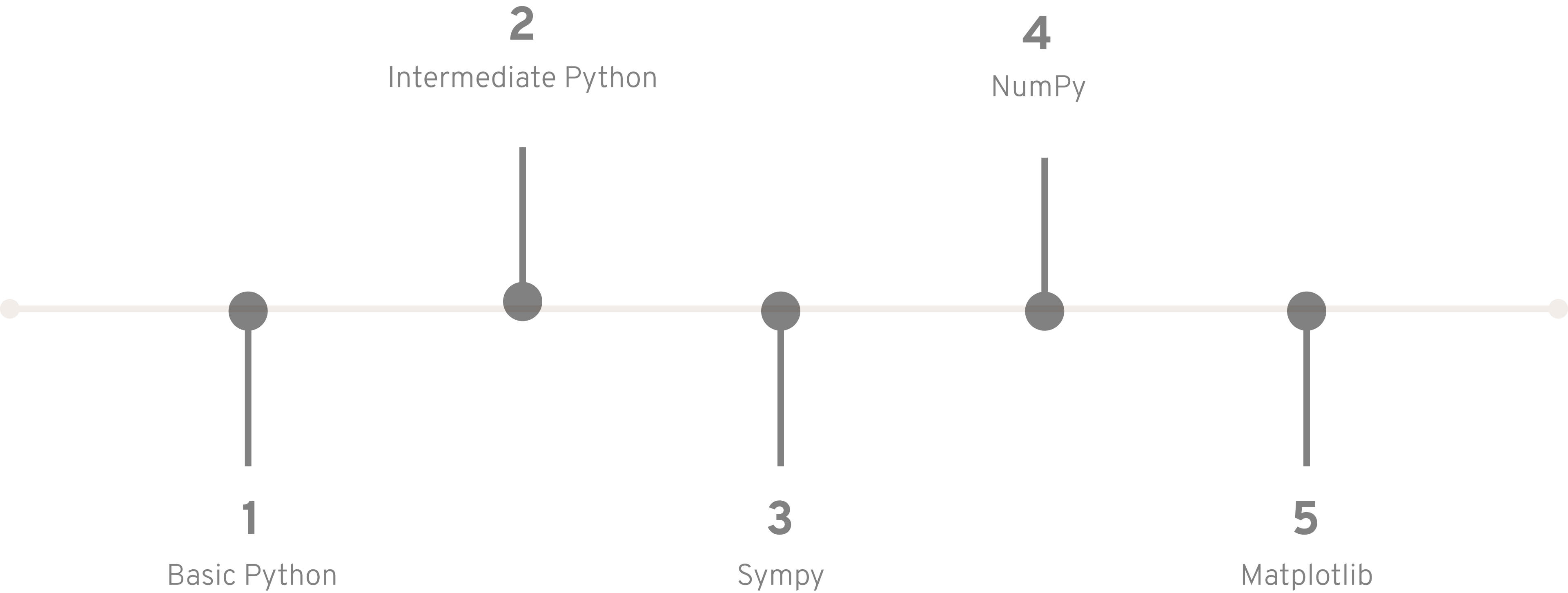
دانشگاه شهید بهشتی ۱۳۳۸

Mathematical Software 2

winter 2024

Maryam Babaei

OUTLINE



1 Basic Python

2 Intermediate Python

3 Sympy

4 NumPy

5 Matplotlib

BASIC PYTHON

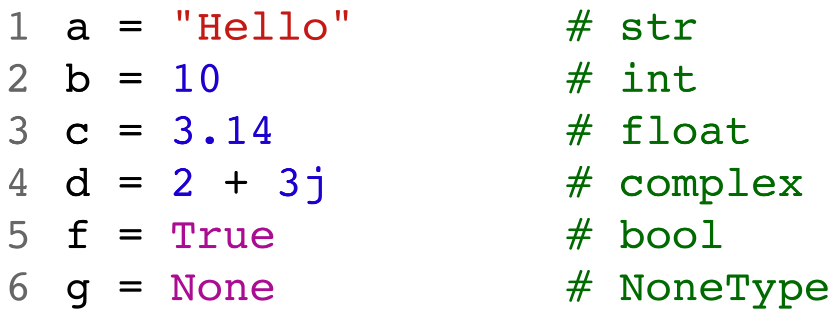
WHAT IS PYTHON?

- An **interpreted**, **high-level**, **general-purpose** programming language
- Dynamic type
- Garbage collection
  - reference counting
- It supports object-oriented and functional programming

INTRODUCTION TO PYTHON

DATA TYPES

- Text Type
  - str
- Numeric Types
  - int, float, ...
- Boolean Type
  - bool
- Void Type
  - None



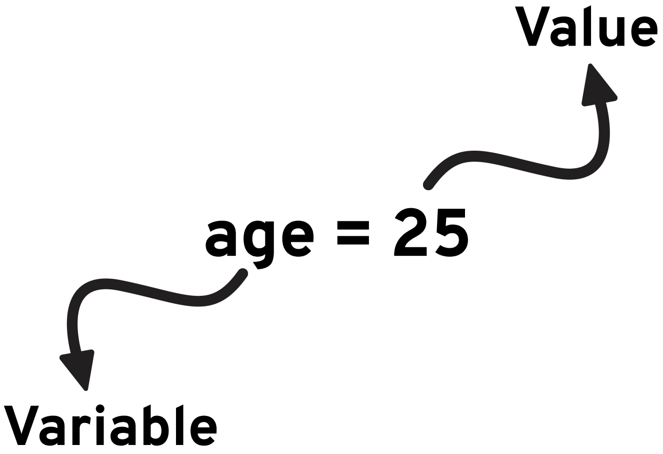
```
a = "Hello"             # str
b = 10                  # int
c = 3.14                # float
d = 2 + 3j              # complex
f = True                # bool
g = None                # NoneType
```

- Comments
  - # symbol for one-line comments
  - """ or ''' For multi-line comment

TYPE CASTING

Function	Conversion
int()	string, float ->int
float()	string, int -> float
str()	int, float - >string
complex()	int, float -> complex

VARIABELS



Value

age = 25

Variable

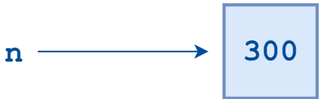
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- Variable names are case-sensitive (age, Age and AGE are three different variables)

RESERVED WORDS

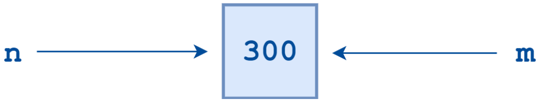
False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

OBJECT REFERENCES

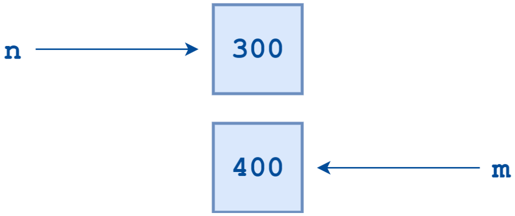
```
>>> n = 300
```



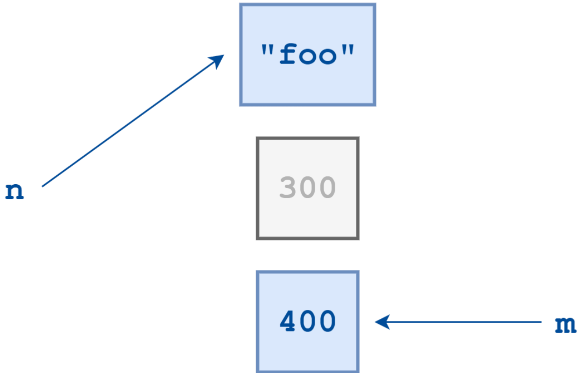
```
>>> m = n
```



```
>>> m = 400
```



```
>>> n = "foo"
```

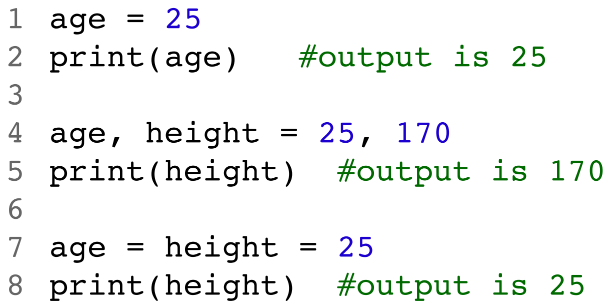


OBJECT IDENTITY

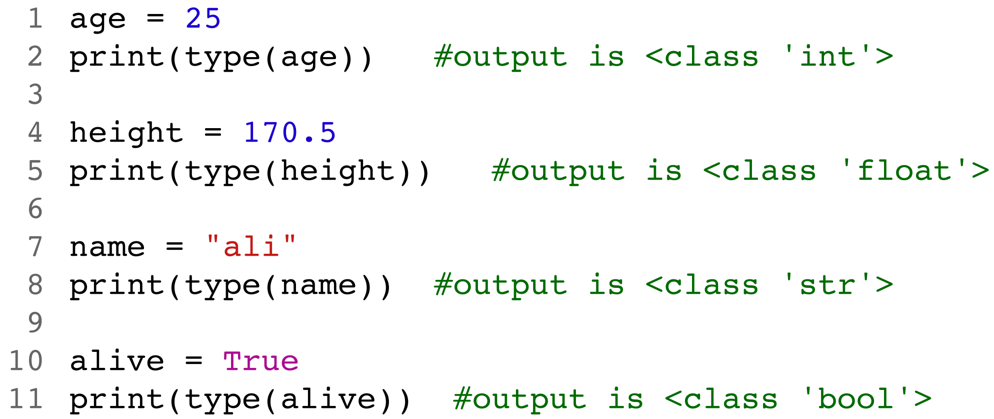
```
>>> n = 300
>>> m = n
>>> id(n)
60127840
>>> id(m)
60127840

>>> m = 400
>>> id(m)
60127872
```

```
>>> m = 300
>>> n = 300
>>> id(m)
60062304
>>> id(n)
60062896

>>> p = 30
>>> q = 30
>>> id(p)
1405569120
>>> id(q)
1405569120
```

BASIC OUTPUT



```python
age = 25
print(age)   #output is 25

age, height = 25, 170
print(height)  #output is 170

age = height = 25
print(height)  #output is 25
```



```python
age = 25
print(type(age))   #output is <class 'int'>

height = 170.5
print(type(height))   #output is <class 'float'>

name = "ali"
print(type(name))  #output is <class 'str'>

alive = True
print(type(alive))  #output is <class 'bool'>
```

BASIC INPUT

```
>>> user_input = input()
foo bar baz
>>> user_input
'foo bar baz'
```

```
>>> number = input("Enter a number: ")
Enter a number: 50
>>> print(number + 100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int

>>> number = int(input("Enter a number: "))
Enter a number: 50
>>> print(number + 100)
150
```

BASIC INPUT

```
a = int(input("Enter a number: "))  #Enter a number: 1
b = float(input("Enter a number: "))  #Enter a number: 1
c = str(input("Enter a number: "))  #Enter a number: 1

print(a, type(a)) #output is 1 <class 'int'>
print(b, type(b)) #output is 1.0 <class 'float'>
print(c, type(c)) #output is 1 <class 'str'>
```

BASIC OPERATIONS

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 OPERATION

Operator	Example	Same As
=	x = 5	x = 5
:=	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 **=2	x = x ** 2

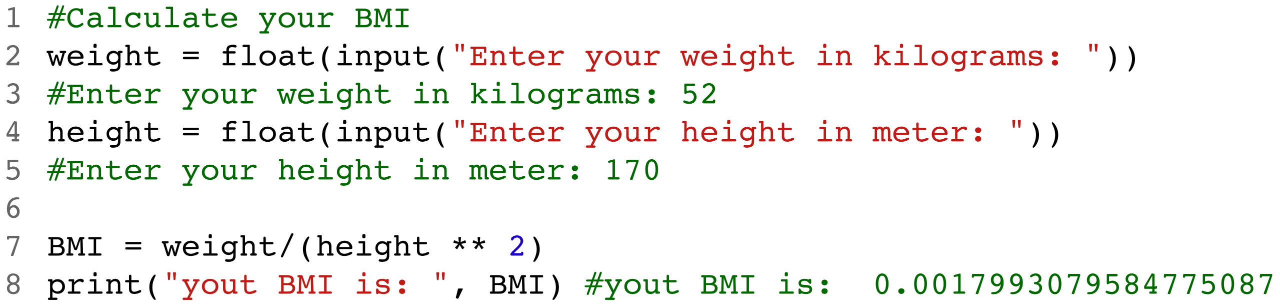
COMPARISON OPERATIONS

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

LOGICAL OPERATIONS

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)

SIMPLE EXAMPLE



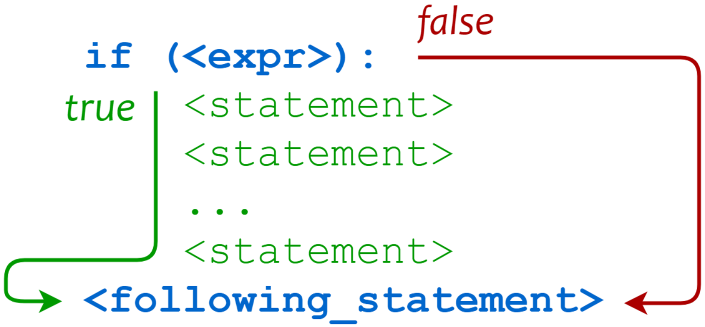
```python
#Calculate your BMI
weight = float(input("Enter your weight in kilograms: "))
#Enter your weight in kilograms: 52
height = float(input("Enter your height in meter: "))
#Enter your height in meter: 170

BMI = weight/(height ** 2)
print("yout BMI is: ", BMI) #yout BMI is:  0.0017993079584775087
```

CONTROL STRUCTURES

CONDITIONAL STATEMENTS

```
if <expr>:
    <statement>
```

```
if <expr>:
    <statement>
    <statement>
    ...
    <statement>
<following_statement>
```



```
if (<expr>):
    <statement>
    <statement>
    ...
    <statement>
<following_statement>
```

CONDITIONAL STATEMENTS

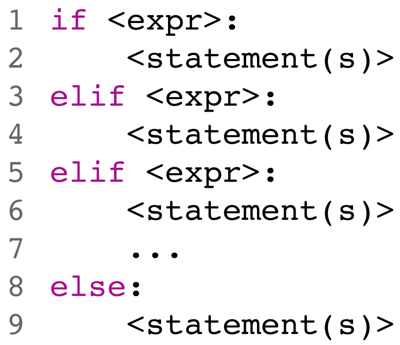
```
>>> x = 0
>>> y = 5

>>> if x < y:                            # Truthy
...     print('yes')                     #output is yes

yes
>>> if y < x:                            # Falsy
...     print('yes')

>>> if y < x or x < y:                   # Truthy
...     print('yes')                     #output is yes

>>> if y < x and x < y:                  # Falsy
...     print('yes')

>>> if 'aul' in 'grault':                # Truthy
...     print('yes')                     #output is yes

```

CONDITIONAL STATEMENTS

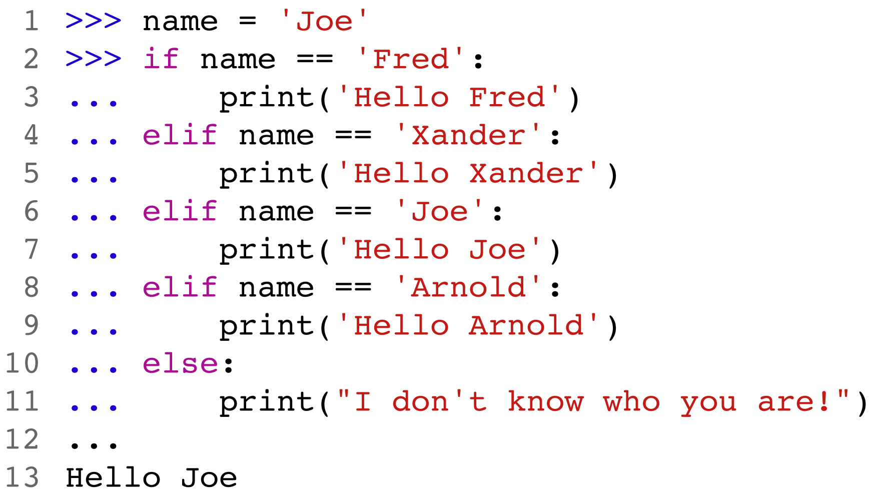
```
if <expr>:
    <statement(s)>
else:
    <statement(s)>
```

```
>>> x = 20

>>> if x < 50:
...     print('(first suite)')
...     print('x is small')
... else:
...     print('(second suite)')
...     print('x is large')
...
(first suite)
x is small
```

CONDITIONAL STATEMENTS

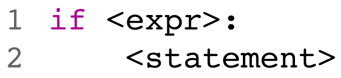


```
if <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
    ...
else:
    <statement(s)>
```

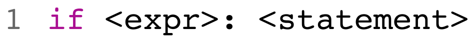


```
>>> name = 'Joe'
>>> if name == 'Fred':
...     print('Hello Fred')
... elif name == 'Xander':
...     print('Hello Xander')
... elif name == 'Joe':
...     print('Hello Joe')
... elif name == 'Arnold':
...     print('Hello Arnold')
... else:
...     print("I don't know who you are!")
...
Hello Joe
```

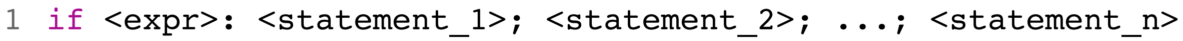
CONDITIONAL STATEMENTS



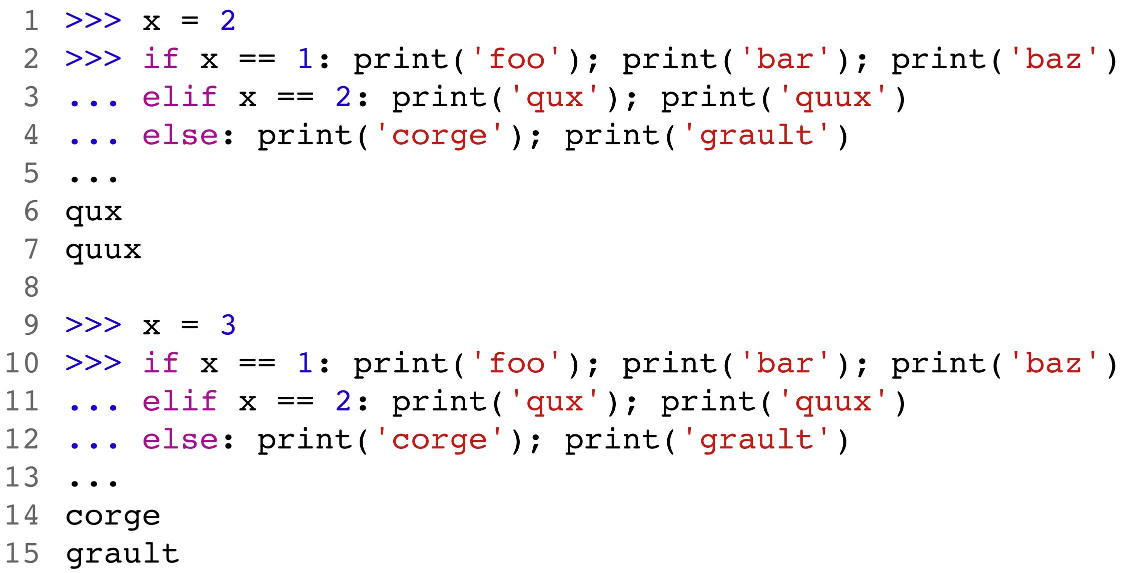
```
if <expr>:
    <statement>
```



```
if <expr>: <statement>
```



```
if <expr>: <statement_1>; <statement_2>; ...; <statement_n>
```



```
>>> x = 2
>>> if x == 1: print('foo'); print('bar'); print('baz')
... elif x == 2: print('qux'); print('quux')
... else: print('corge'); print('grault')
...
qux
quux

>>> x = 3
>>> if x == 1: print('foo'); print('bar'); print('baz')
... elif x == 2: print('qux'); print('quux')
... else: print('corge'); print('grault')
...
corge
grault
```

CONDITIONAL STATEMENTS

```
if <expr>:
    <statement(s)>
else:
    <statement(s)>
```

```
<expr1> if <conditional_expr> else <expr2>
```

```
>>> m = a if a > b else b
```

```
>>> x = y = 40

>>> z = 1 + x if x > y else y + 2
>>> z
42

>>> z = (1 + x) if x > y else (y + 2)
>>> z
42
```

LOOP!

```
for i in <collection>:
    <loop body>
```

```
for <var> in <iterable>:
    <statement(s)>
```

```
>>> for n in (0, 1, 2, 3):
...     print(n)
...
0
1
2
3
```

```
>>> for n in range(0, 4):
...     print(n)
...
0
1
2
3
```

LOOP!

```
>>> for n in range(0, 6, 2):
...     print(n)
...
0
2
4
```

```
>>> for n in range(4, 0, -1):
...     print(n)
...
4
3
2
1
```

LOOP!

```
while <expr>:
    <statement(s)>
```

```
>>> n = 5
>>> while n > 0:
...     n -= 1
...     print(n)
...
4
3
2
1
0
```

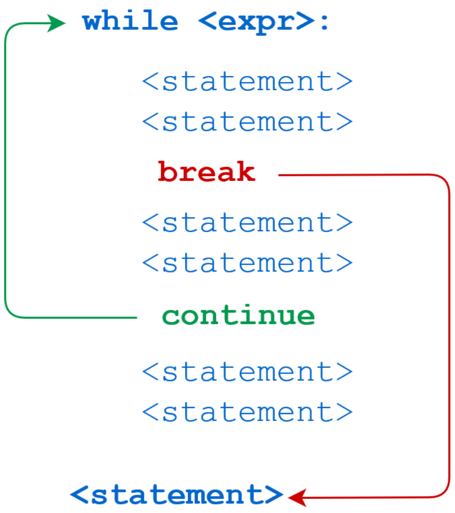
```
>>> n = 0
>>> while n > 0:
...     n -= 1
...     print(n)
...
```

LOOP!

```
while <expr>: <statement>
```

```
>>> n = 5
>>> while n > 0: n -= 1; print(n)

4
3
2
1
0
```

LOOP!



```
while <expr>:
    <statement>
    <statement>
    break
    <statement>
    <statement>
    continue
    <statement>
    <statement>
<statement>
```

```
>>>n = 5
>>>while n > 0:
...     n -= 1
...     if n == 2:
...         break
...     print(n)
>>>print('Loop ended.')
4
3
Loop ended.
```

```
>>>n = 5
>>>while n > 0:
...     n -= 1
...     if n == 2:
...         continue
...     print(n)
>>>print('Loop ended.')
4
3
1
0
Loop ended.
```

LOOP!

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

```
>>> n = 5
>>> while n > 0:
...     n -= 1
...     print(n)
... else:
...     print('Loop done.')
...
4
3
2
1
0
Loop done.
```

LOOP!

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

```
>>> n = 5
>>> while n > 0:
...     n -= 1
...     print(n)
...     if n == 2:
...         break
... else:
...     print('Loop done.')
...
4
3
2
```

NESTED WHILE LOOPS!

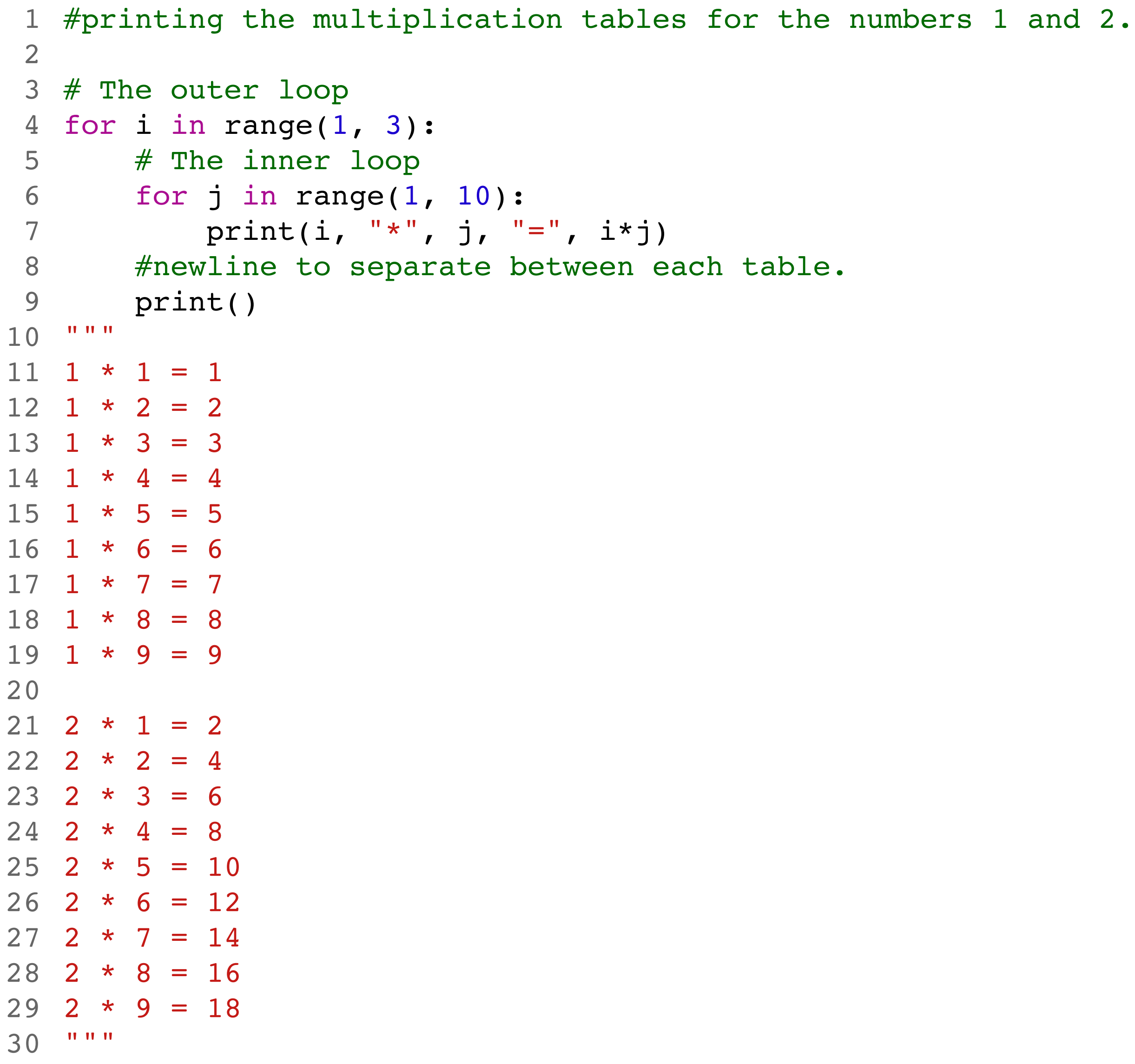
```
while <expr1>:
    statement
    statement

    while <expr2>:
        statement
        statement
        break  # Applies to while <expr2>: loop

    break  # Applies to while <expr1>: loop
```

```
if <expr>:
    statement
    while <expr>:
        statement
        statement
else:
    while <expr>:
        statement
        statement
    statement
```

NESTED WHILE LOOPS!

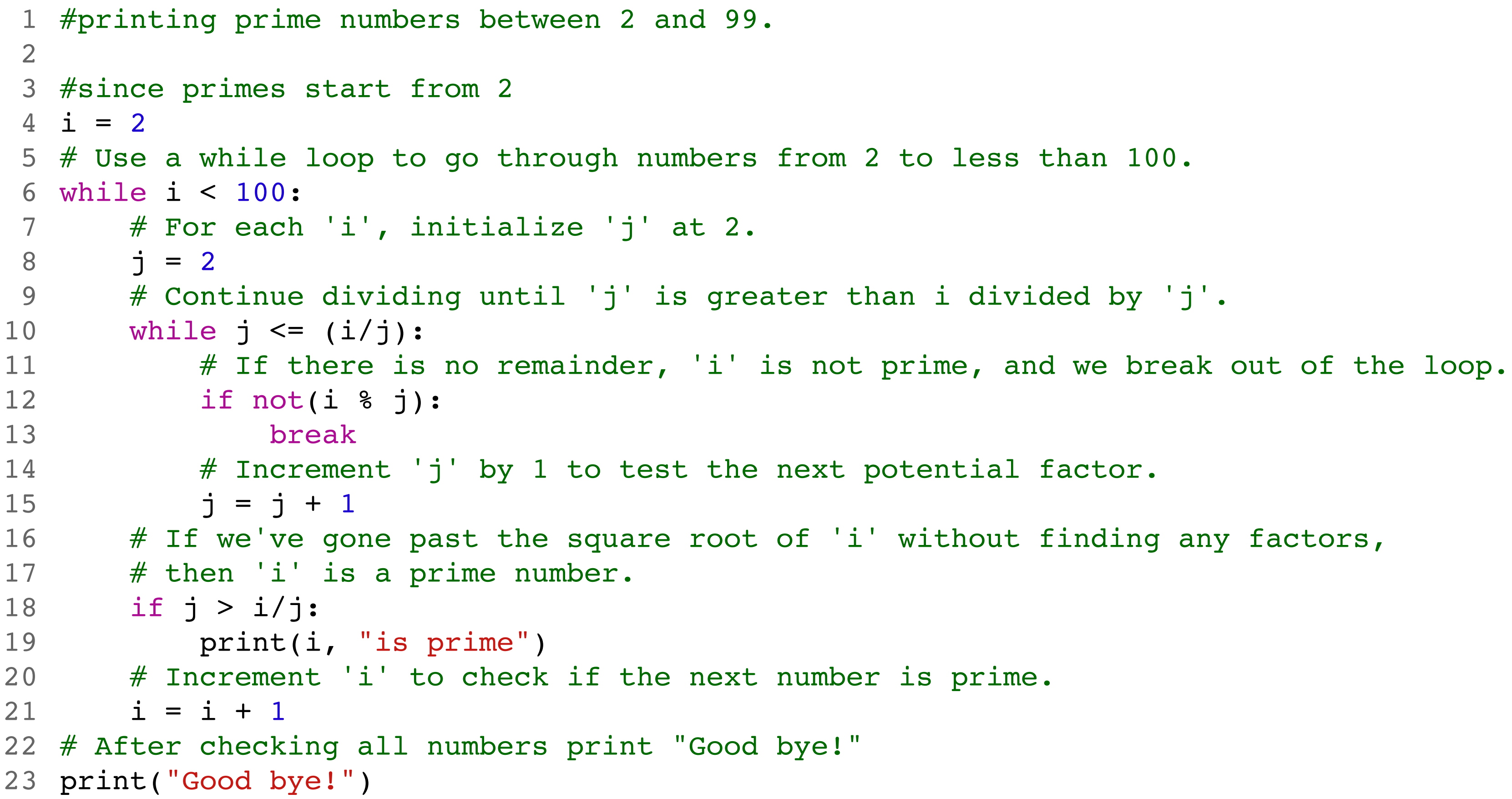
```
while <expr>:
    if <expr>:
        statement
    elif <expr>:
        statement
    else:
        statement

    if <expr>:
        statement
```

EXAMLE



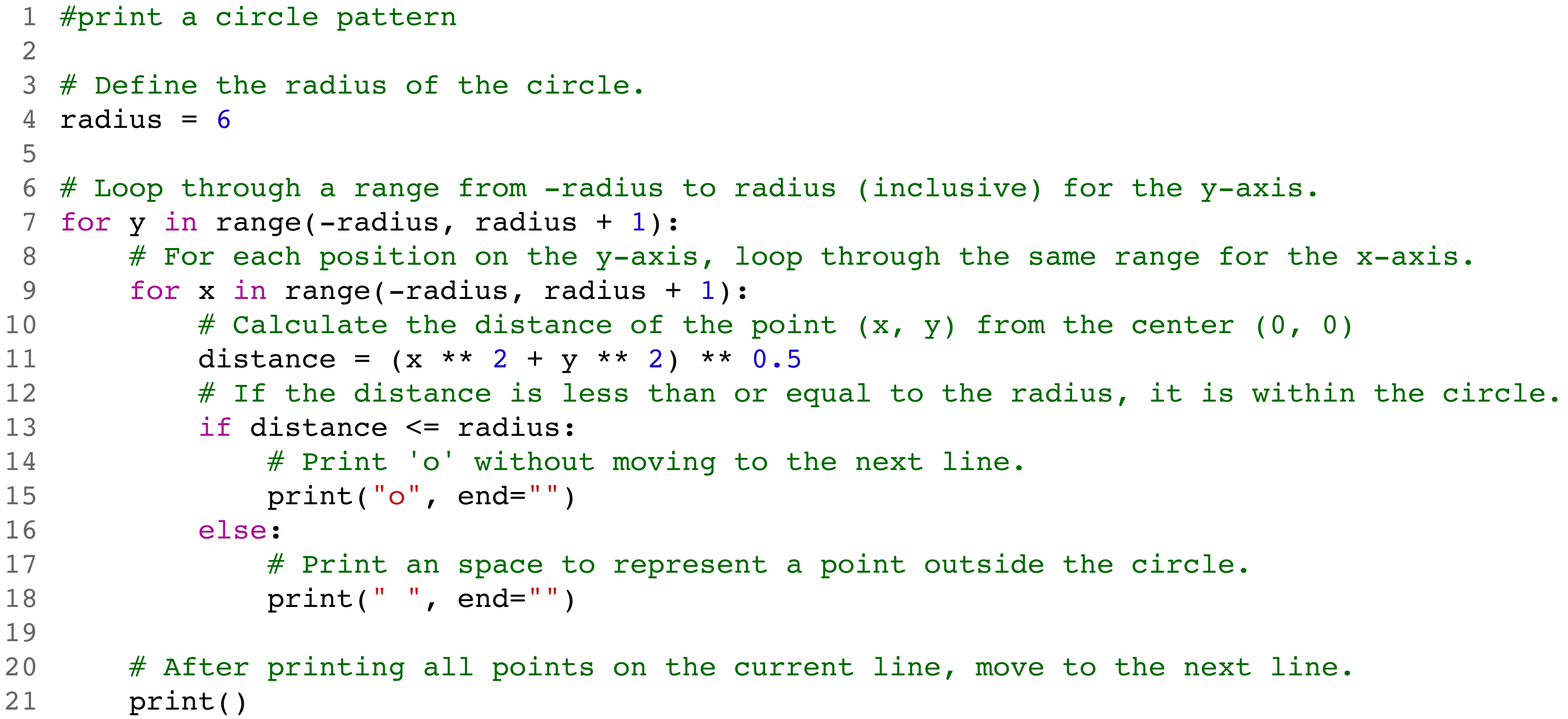
```
#printing the multiplication tables for the numbers 1 and 2.

# The outer loop
for i in range(1, 3):
    # The inner loop
    for j in range(1, 10):
        print(i, "*", j, "=", i*j)
    #newline to separate between each table.
    print()
"""
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
"""
```

EXAMLE



```python
#printing prime numbers between 2 and 99.

#since primes start from 2
i = 2
# Use a while loop to go through numbers from 2 to less than 100.
while i < 100:
    # For each 'i', initialize 'j' at 2.
    j = 2
    # Continue dividing until 'j' is greater than i divided by 'j'.
    while j <= (i/j):
        # If there is no remainder, 'i' is not prime, and we break out of the loop.
        if not(i % j):
            break
        # Increment 'j' by 1 to test the next potential factor.
        j = j + 1
    # If we've gone past the square root of 'i' without finding any factors,
    # then 'i' is a prime number.
    if j > i/j:
        print(i, "is prime")
    # Increment 'i' to check if the next number is prime.
    i = i + 1
# After checking all numbers print "Good bye!"
print("Good bye!")
```

EXAMLE



```python
#print a circle pattern

# Define the radius of the circle.
radius = 6

# Loop through a range from -radius to radius (inclusive) for the y-axis.
for y in range(-radius, radius + 1):
    # For each position on the y-axis, loop through the same range for the x-axis.
    for x in range(-radius, radius + 1):
        # Calculate the distance of the point (x, y) from the center (0, 0)
        distance = (x ** 2 + y ** 2) ** 0.5
        # If the distance is less than or equal to the radius, it is within the circle.
        if distance <= radius:
            # Print 'o' without moving to the next line.
            print("o", end="")
        else:
            # Print an space to represent a point outside the circle.
            print(" ", end="")

    # After printing all points on the current line, move to the next line.
    print()
```

INTERMEDIATE PYTHON

FUNCTIONS

BUILT-IN FUNCTIONS

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

BUILT-IN FUNCTIONS

```
>>>pow(2, 3)
8

>>>pow(2, 3, mod=3)
2
>>>2**3 % 3 == 2
True

>>>round(4.5)
4

>>>max(3, 4, 1)
4

>>len("hello")
5
```

USER-DEFINED FUNCTIONS

mathematical concept of a function

$$z = f(x,y)$$

```
def <function_name>([<parameters>]):
    <statement(s)>
```

USER-DEFINED FUNCTIONS

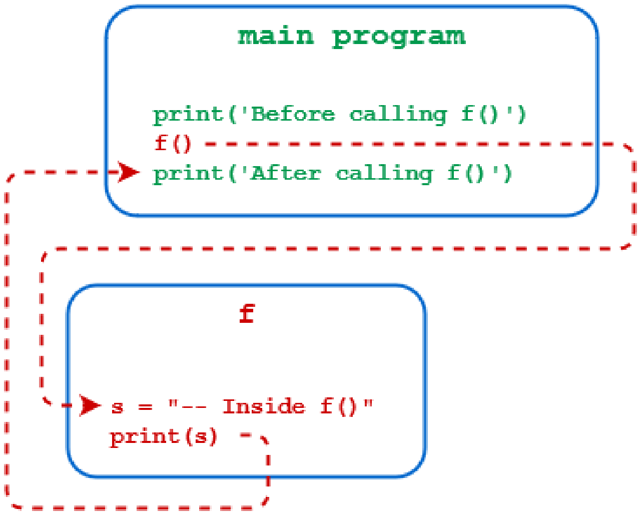
```
def <function_name>([<parameters>]):
    <statement(s)>
```

```
<function_name>([<arguments>])
```

Component	**Meaning**
def	The keyword that informs Python that a function is being defined
<function_name>	A valid Python identifier that names the function
<parameters>	An optional, comma-separated list of parameters that may be passed to the function
:	Punctuation that denotes the end of the Python function header (the name and parameter list)
<statement(s)>	A block of valid Python statements

USER-DEFINED FUNCTIONS

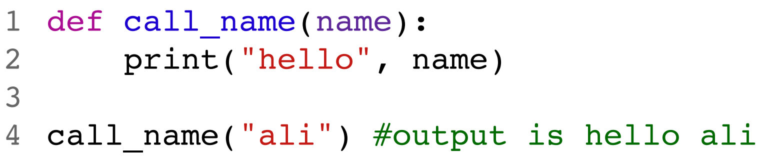
```python
def f():
    s = '-- Inside f()'
    print(s)

print('Before calling f()')
f()
print('After calling f()')
```



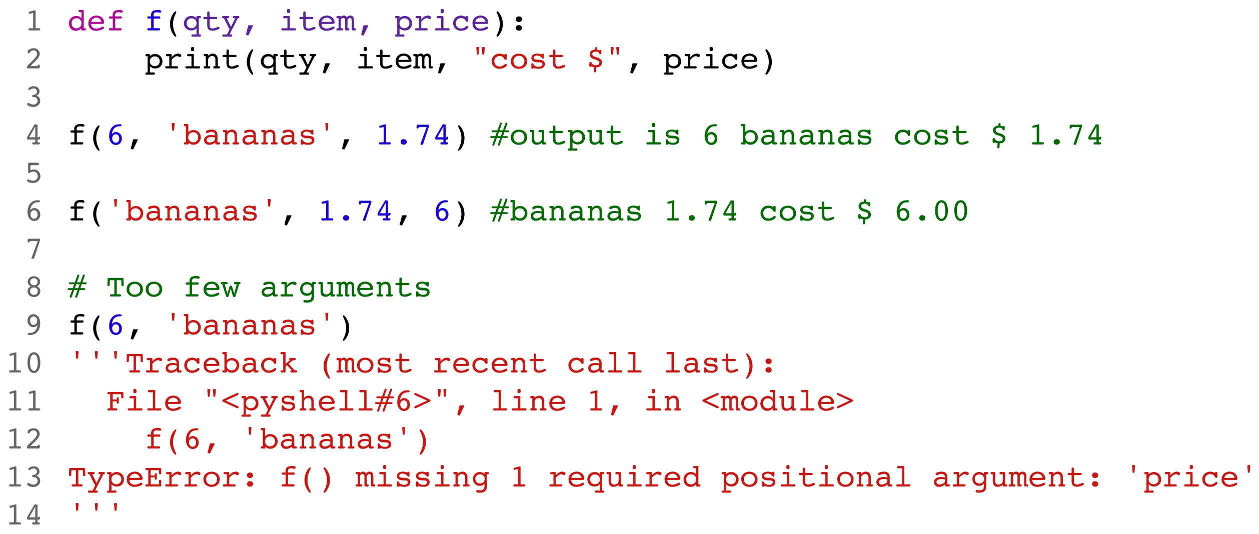
```python
print('Before calling f()')
f()
print('After calling f()')
```

```python
s = "-- Inside f()"
print(s)
```

USER-DEFINED FUNCTIONS



```python
def call_name(name):
    print("hello", name)

call_name("ali") #output is hello ali
```



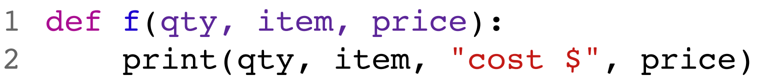
```python
def f(qty, item, price):
    print(qty, item, "cost $", price)

f(6, 'bananas', 1.74) #output is 6 bananas cost $ 1.74

f('bananas', 1.74, 6) #bananas 1.74 cost $ 6.00

# Too few arguments
f(6, 'bananas')
'''Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    f(6, 'bananas')
TypeError: f() missing 1 required positional argument: 'price'
'''
```

USER-DEFINED FUNCTIONS

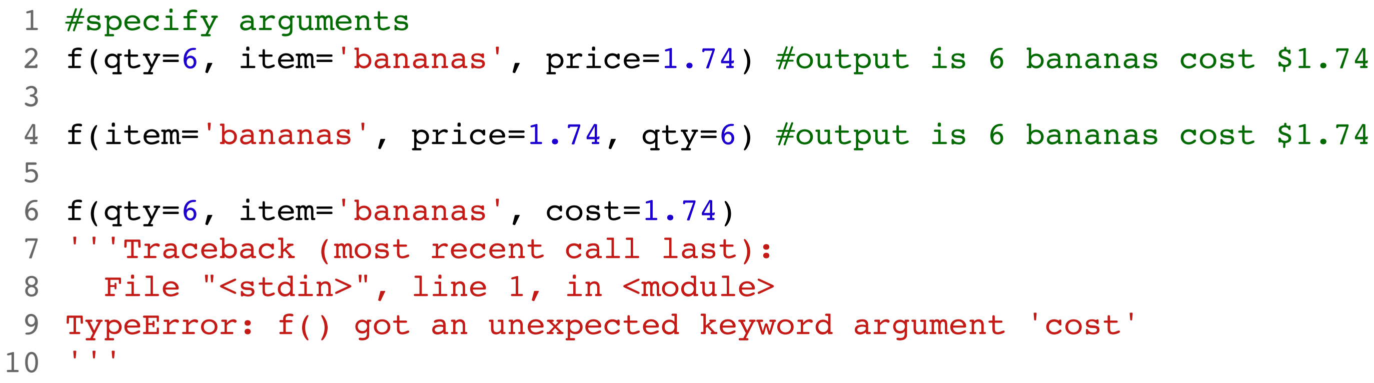
```python
def f(qty, item, price):
    print(qty, item, "cost $", price)
```

```python
# Too few arguments
f(6, 'bananas')
'''Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    f(6, 'bananas')
TypeError: f() missing 1 required positional argument: 'price'
'''

# Too many arguments
f(6, 'bananas', 1.74, 'kumquats')
'''Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    f(6, 'bananas', 1.74, 'kumquats')
TypeError: f() takes 3 positional arguments but 4 were given
'''
```

USER-DEFINED FUNCTIONS



```python
def f(qty, item, price):
    print(qty, item, "cost $", price)
```



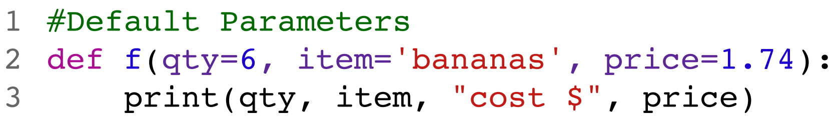
```python
#specify arguments
f(qty=6, item='bananas', price=1.74) #output is 6 bananas cost $1.74

f(item='bananas', price=1.74, qty=6) #output is 6 bananas cost $1.74

f(qty=6, item='bananas', cost=1.74)
'''Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: f() got an unexpected keyword argument 'cost'
'''
```

USER-DEFINED FUNCTIONS

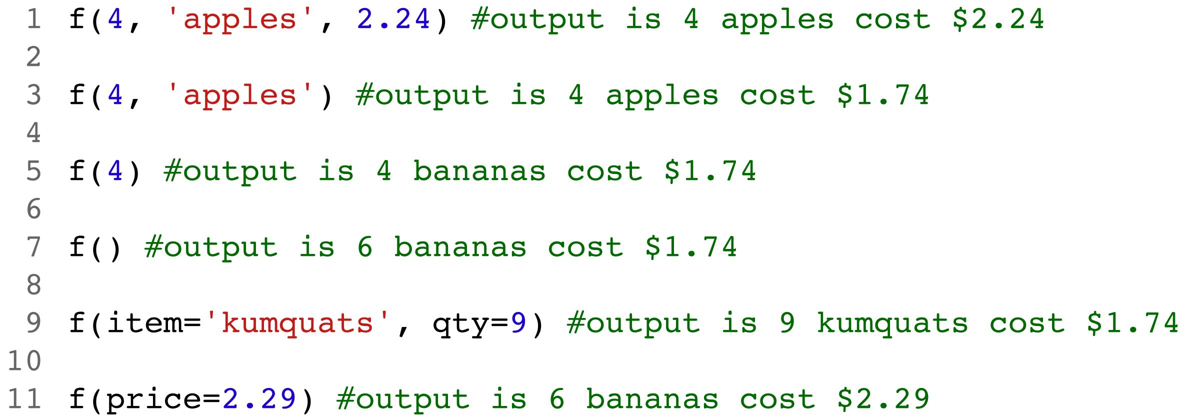
```python
def f(qty, item, price):
    print(qty, item, "cost $", price)
```

```python
f(6, price=1.74, item='bananas') #output is 6 bananas cost $1.74

f(6, 'bananas', price=1.74) #output is 6 bananas cost $1.74

f(6, item='bananas', 1.74)
#SyntaxError: positional argument follows keyword argument
```

USER-DEFINED FUNCTIONS

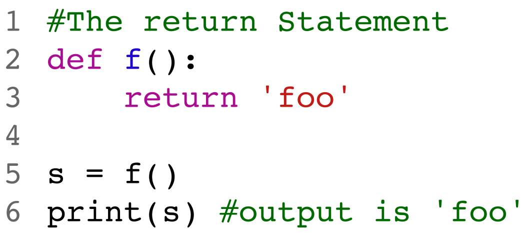


```python
#Default Parameters
def f(qty=6, item='bananas', price=1.74):
    print(qty, item, "cost $", price)
```

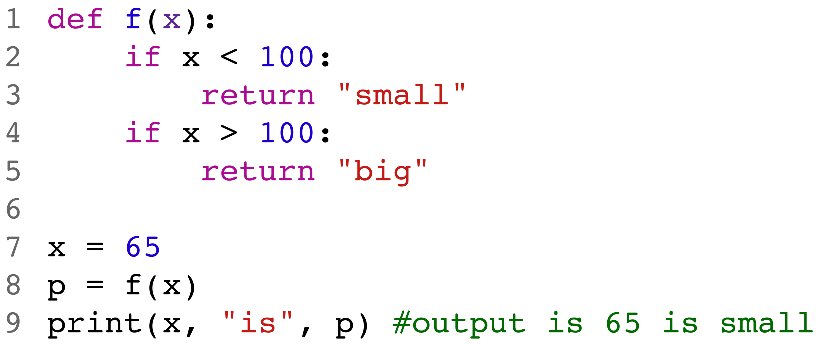


```python
f(4, 'apples', 2.24) #output is 4 apples cost $2.24

f(4, 'apples') #output is 4 apples cost $1.74

f(4) #output is 4 bananas cost $1.74

f() #output is 6 bananas cost $1.74

f(item='kumquats', qty=9) #output is 9 kumquats cost $1.74

f(price=2.29) #output is 6 bananas cost $2.29
```

USER-DEFINED FUNCTIONS

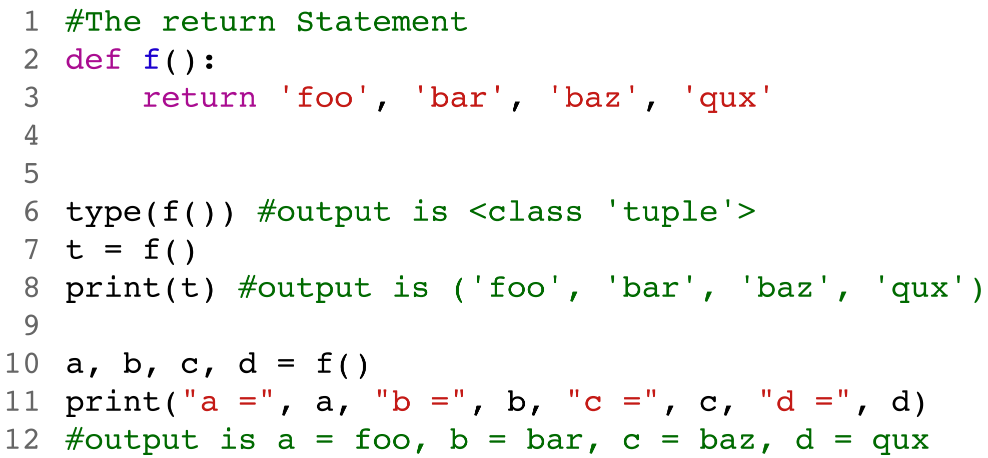


```python
#The return Statement
def f():
    return 'foo'

s = f()
print(s) #output is 'foo'
```



```python
def f(x):
    if x < 100:
        return "small"
    if x > 100:
        return "big"

x = 65
p = f(x)
print(x, "is", p) #output is 65 is small
```

USER-DEFINED FUNCTIONS



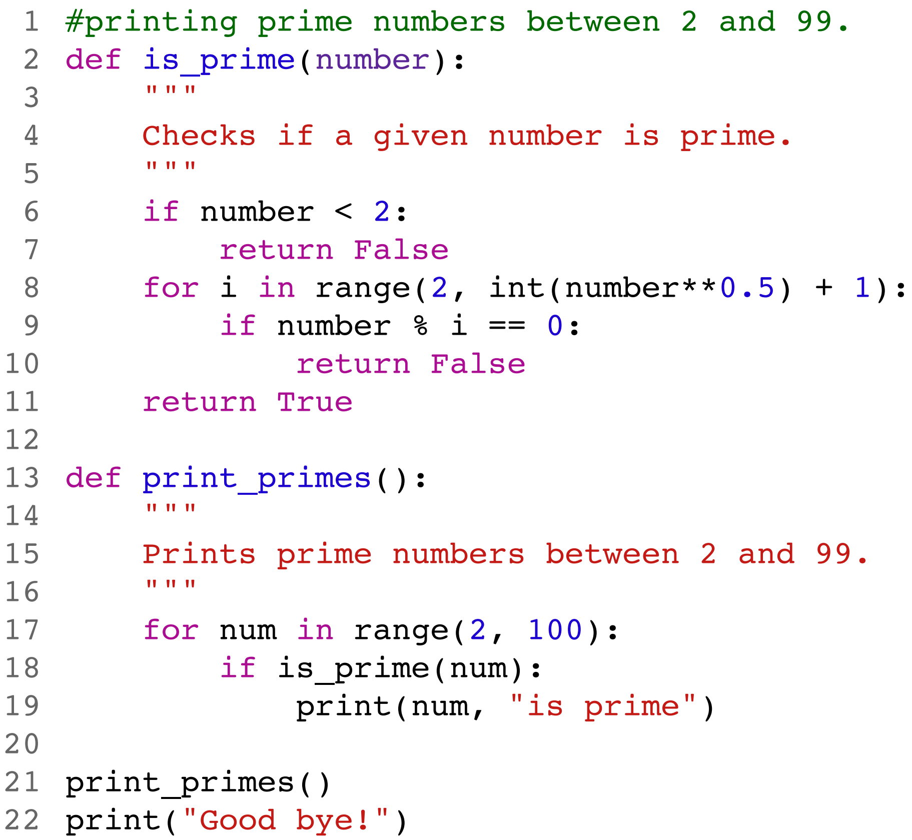
```python
#The return Statement
def f():
    return 'foo', 'bar', 'baz', 'qux'


type(f()) #output is <class 'tuple'>
t = f()
print(t) #output is ('foo', 'bar', 'baz', 'qux')

a, b, c, d = f()
print("a =", a, "b =", b, "c =", c, "d =", d)
#output is a = foo, b = bar, c = baz, d = qux
```

USER-DEFINED FUNCTIONS

```
def double(x):
    return x * 2


x = 5
x = double(x)
print(x) #output is 10
```

```
def avg(a, b, c):
    return (a + b + c) / 3

print(avg(1, 2, 3)) #output is 2.0
```

EXAMPLE



```python
#printing prime numbers between 2 and 99.
def is_prime(number):
    """
    Checks if a given number is prime.
    """
    if number < 2:
        return False
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return False
    return True

def print_primes():
    """
    Prints prime numbers between 2 and 99.
    """
    for num in range(2, 100):
        if is_prime(num):
            print(num, "is prime")

print_primes()
print("Good bye!")
```

USER-DEFINED FUNCTIONS

```
#Argument Tuple Packing
def f(*args):
    print(args)
    for x in args:
        print(x)
```

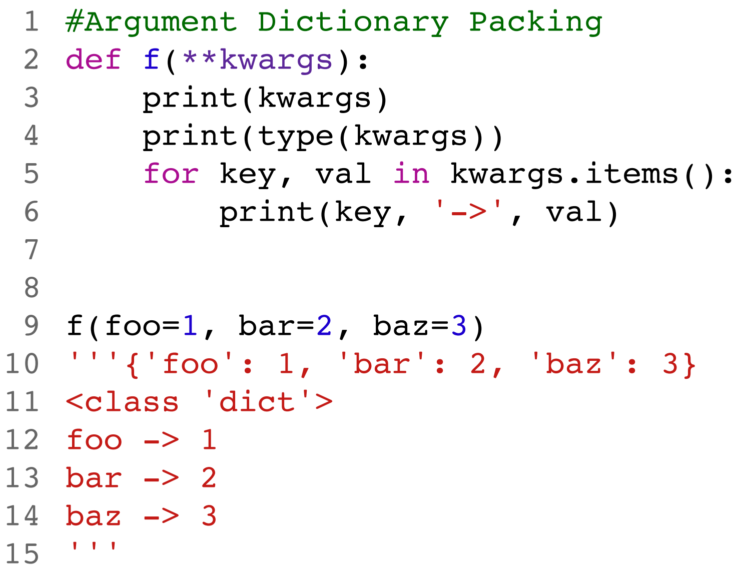
```
f(1, 2, 3)
'''(1, 2, 3)
<class 'tuple'> 3
1
2
3
'''

f('foo', 'bar', 'baz', 'qux', 'quux')
'''('foo', 'bar', 'baz', 'qux', 'quux')
<class 'tuple'> 5
foo
bar
baz
qux
quux
'''
```

USER-DEFINED FUNCTIONS

```python
#Argument Tuple Packing
def avg(*args):
    total = 0
    for i in args:
        total += i
    return total / len(args)


print(avg(1, 2, 3)) #output is 2.0
print(avg(1, 2, 3, 4, 5)) #output is 3.0
```

```python
def avg(*args):
    return sum(args) / len(args)


print(avg(1, 2, 3)) #output is 2.0
print(avg(1, 2, 3, 4, 5)) #output is 3.0
```

USER-DEFINED FUNCTIONS



```python
#Argument Dictionary Packing
def f(**kwargs):
    print(kwargs)
    print(type(kwargs))
    for key, val in kwargs.items():
        print(key, '->', val)


f(foo=1, bar=2, baz=3)
'''{'foo': 1, 'bar': 2, 'baz': 3}
<class 'dict'>
foo -> 1
bar -> 2
baz -> 3
'''
```

USER-DEFINED FUNCTIONS

```
#Argument Dictionary Packing
def f(a, b, *args, **kwargs):
    print(F'a = {a}')
    print(F'b = {b}')
    print(F'args = {args}')
    print(F'kwargs = {kwargs}')


f(1, 2, 'foo', 'bar', 'baz', 'qux', x=100, y=200, z=300)
'''a = 1
b = 2
args = ('foo', 'bar', 'baz', 'qux')
kwargs = {'x': 100, 'y': 200, 'z': 300}
'''
```

DATA STRUCTURES

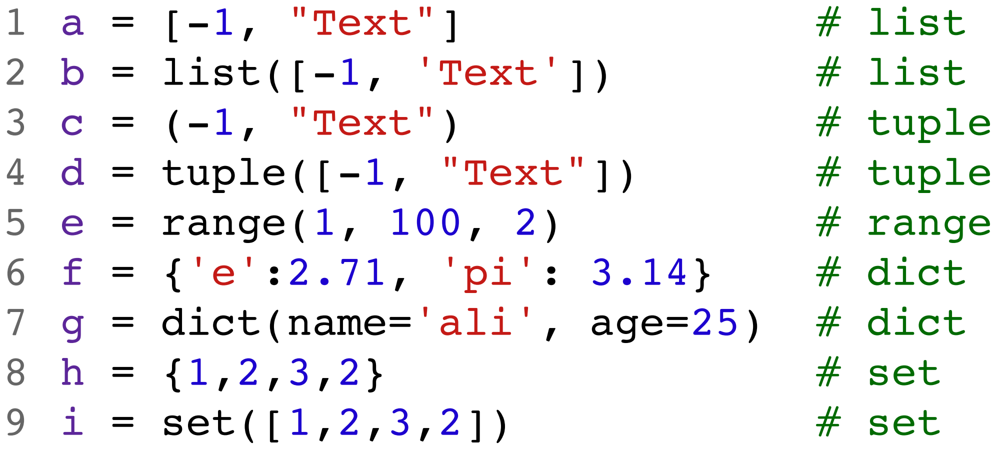
DATA TYPES

- Text Type
  - str
- Numeric Types
  - int, float, complex
- Boolean Type
  - bool
- Void Type
  - None

```python
a = "Hello"        # str
b = 'Hello'        # str
c = str(10)        # str
d = 10             # int
e = int(3.1)       # int
f = 3.14           # float
g = float('1.4')   # float
h = 2 + 3j         # complex
i = complex(2,3)   # complex
j = True           # bool
k = False          # bool
l = bool(1)        # bool
m = None           # NoneType
```

DATA TYPES

- Sequence Types
  - list, tuple, range
- Mapping Type
  - dict
- Set Types
  - set
- Binary Types
  - bytes



```
a = [-1, "Text"]                # list
b = list([-1, 'Text'])          # list
c = (-1, "Text")                # tuple
d = tuple([-1, "Text"])         # tuple
e = range(1, 100, 2)            # range
f = {'e':2.71, 'pi': 3.14}      # dict
g = dict(name='ali', age=25)    # dict
h = {1,2,3,2}                   # set
i = set([1,2,3,2])              # set
```

STRINGS

```
>>> s = 'foo'
>>> t = 'bar'
>>> u = 'baz'

>>> s + t
'foobar'
>>> s + t + u
'foobarbaz'

>>> print('Go team' + '!!!')
Go team!!!
```

```
>>> s = 'foo.'

>>> s * 4
'foo.foo.foo.foo.'
>>> 4 * s
'foo.foo.foo.foo.'
>>> 'foo' * -8
''
```

STRINGS

```
>>> s = 'foo'

>>> s in 'That\'s food for thought.'
True
>>> s in 'That\'s good for now.'
False
```

```
>>> 'z' not in 'abc'
True
>>> 'z' not in 'xyz'
False
```

STRINGS

```
>>> s = 'I am a string.'
>>> len(s)
14
```

```
>>> str(49.2)
'49.2'
>>> str(3+4j)
'(3+4j)'
>>> str(3 + 29)
'32'
>>> str('foo')
'foo'
```

STRINGS INDEXING

```
>>> s = 'foobar'

>>> s[0]
'f'
>>> s[1]
'o'
>>> s[3]
'b'
>>> len(s)
6
>>> s[len(s)-1]
'r'
>>> s = 'foobar'
>>> s[-1]
'r'
>>> s[-2]
'a'
>>> len(s)
6
>>> s[-len(s)]
'f'
```

STRINGS INDEXING

```
>>> s = 'foobar'
>>> s[2:5]
'oba'
>>> s = 'foobar'
>>> s[:4]
'foob'
>>> s[0:4]
'foob'
>>> s = 'foobar'
>>> s[2:]
'obar'
>>> s[2:len(s)]
'obar'
>>> s = 'foobar'
>>> s[:4] + s[4:]
'foobar'
>>> s[:4] + s[4:] == s
True
>>> s = 'foobar'
>>> t = s[:]
>>> s is t
True
```

STRINGS INDEXING

```
>>> s = 'foobar'

>>> s[0:6:2]
'foa'

>>> s[1:6:2]
'obr'

>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::5]
'11111'
>>> s[4::5]
'55555'
>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::-5]
'55555'
>>> s = 'If Comrade Napoleon says it, it must be right.'
>>> s[::-1]
'.thgir eb tsum ti ,ti syas noelopaN edarmoC fI'
```

STRINGS

```
>> 'hello ali'.split()
['hello', 'ali']

>> 'Hello ali'.replace('Hello', 'Bye')
'Bye ali'

>> '-'.join(['a', 'b', 'c'])
'a-b-c'

>> 'Hello'.upper()
'HELLO'

>> 'Hello'.lower()
'hello'

>>> 'foo bar foo baz foo qux'.find('foo')
0
>>> 'foo bar foo baz foo qux'.find('foo', 4)
8
```

STRINGS

Method	Description
capitalize()	Converts the first character to upper case
count()	Returns the number of times a specified value occurs in a string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
join()	Converts the elements of an iterable into a string
lower()	Converts a string into lower case
replace()	Returns a string where a specified value is replaced with a specified value
split()	Splits the string at the specified separator, and returns a list

LISTS

```
>>> colors = [
...     "red",
...     "orange",
...     "yellow",
...     "green",
...     "blue",
...     "indigo",
...     "violet"
... ]

>>> colors
['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
```

LISTS

```
>>> colors[0]
'red'
>>> colors[1]
'orange'
>>> colors[2]
'yellow'
>>> colors[3]
'green'

>>> colors[-1]
'green'
>>> colors[-2]
'yellow'
>>> colors[-3]
'orange'
>>> colors[-4]
'red'

>>> languages[-7]
Traceback (most recent call last):
    ...
IndexError: list index out of range
```

LISTS

```
>>> colors[0:2]
['red', 'orange']
>>> colors[1:]
['orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
>>> colors[0::2]
['red', 'yellow', 'blue', 'violet']
>>> colors[0::2]
['red', 'yellow', 'blue', 'violet']
>>> colors[-8::1]
['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
```

LISTS

```
>>> countries = ["United States", "Canada", "Poland", "Germany", "Austria"]

>>> nations = countries
>>> id(countries) == id(nations)
True

>>> nations = countries[:]
>>> nations
['United States', 'Canada', 'Poland', 'Germany', 'Austria']

>>> id(countries) == id(nations)
False

>>>from copy import copy
>>> nations = countries.copy()
>>> nations
['United States', 'Canada', 'Poland', 'Germany', 'Austria']

>>> id(countries) == id(nations)
False
```

LISTS

```
>>> pets = ["cat", "dog"]

>>> pets.append("parrot")
['cat', 'dog', 'parrot']

>>> pets.append(['hamster', 'turtle'])
['cat', 'dog', 'parrot', ['hamster', 'turtle']]


>>> pets.extend(['hamster', 'turtle'])
['cat', 'dog', 'parrot', 'hamster', 'turtle']

>>> pets.insert(2, 'hamster')
['cat', 'dog', 'hamster', 'parrot', 'hamster', 'turtle']

>>> pets.remove('hamster')
['cat', 'dog', 'parrot', 'hamster', 'turtle']

>>> visited = pets.pop()
>>> visited
'turtle'
>>> pets
['cat', 'dog', 'parrot', 'hamster']
```

LISTS

```
>>> numbers = ["2", "9", "5", "1", "6"]

>>> for i, number in enumerate(numbers):
...     numbers[i] = int(number)
...

>>> numbers
[2, 9, 5, 1, 6]
```

LISTS

Method	**Description**
append()	Adds an element at the end 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
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 first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

TUPLES

```
>>> record = ("John", 35, "Python Developer")

>>> record[0] = "Adel"
TypeError: 'tuple' object does not support item assignment

>>> record[0]
'John'
>>> record[1]
35
>>> record[2]
'Python Developer'

>>> record[:2]
("John", 3)

>>> record[1:]
(35, "Python Developer")
```

TUPLES

```
>>> point = (7, 14, 21)

>>> x, y, z = point
>>> x
7
>>> y
14
>>> z
21
```

TUPLES

```
>>> student_info = ("Linda", 18, ["Math", "Physics", "History"])

>>> student_profile = student_info[:]
>>> id(student_info) == id(student_profile)
True

>>> from copy import copy
>>> student_info = ("Linda", 18, ["Math", "Physics", "History"])
>>> student_profile = copy(student_info)
>>> id(student_info) == id(student_profile)
True
```

SETS

```
>>> x = set(['foo', 'bar', 'baz', 'foo', 'qux'])
>>> x
{'qux', 'foo', 'bar', 'baz'}

>>> x = set(('foo', 'bar', 'baz', 'foo'))
>>> x
{'foo', 'bar', 'baz'}

>>> len(x)
3

>>> 'bar' in x
True
>>> 'qux' in x
False
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}
>>> x1 | x2
{'baz', 'quux', 'qux', 'bar', 'foo'}

>>> x1.union(x2)
{'baz', 'quux', 'qux', 'bar', 'foo'}

>>> a = {1, 2, 3, 4}
>>> b = {2, 3, 4, 5}
>>> c = {3, 4, 5, 6}
>>> d = {4, 5, 6, 7}

>>> a.union(b, c, d)
{1, 2, 3, 4, 5, 6, 7}

>>> a | b | c | d
{1, 2, 3, 4, 5, 6, 7}
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}

>>> x1.intersection(x2)
{'baz'}

>>> x1 & x2
{'baz'}

>>> a = {1, 2, 3, 4}
>>> b = {2, 3, 4, 5}
>>> c = {3, 4, 5, 6}
>>> d = {4, 5, 6, 7}

>>> a.intersection(b, c, d)
{4}

>>> a & b & c & d
{4}
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}

>>> x1.difference(x2)
{'foo', 'bar'}

>>> x1 - x2
{'foo', 'bar'}

>>> a = {1, 2, 3, 30, 300}
>>> b = {10, 20, 30, 40}
>>> c = {100, 200, 300, 400}

>>> a.difference(b, c)
{1, 2, 3}

>>> a - b - c
{1, 2, 3}
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}

>>> x1.symmetric_difference(x2)
{'foo', 'qux', 'quux', 'bar'}

>>> x1 ^ x2
{'foo', 'qux', 'quux', 'bar'}

>>> a = {1, 2, 3, 4, 5}
>>> b = {10, 2, 3, 4, 50}
>>> c = {1, 50, 100}

>>> a ^ b ^ c
{100, 5, 10}
```

SETS

```
>>> x1 = {1, 3, 5}
>>> x2 = {2, 4, 6}

>>> x1.isdisjoint(x2)
True
>>> x1 & x2
set()

>>> x1 = {1, 3, 5}
>>> x2 = {1, 2, 3, 4, 5}
>>> x1.issubset(x2)
True

>>> x1 <= x2
True

>>> x2.issuperset(x1)
True

>>> x2 >= x1
True
```

SETS

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'foo', 'baz', 'qux'}

>>> x1 |= x2
>>> x1
{'qux', 'foo', 'bar', 'baz'}

>>> x1.update(['corge', 'garply'])
>>> x1
{'qux', 'corge', 'garply', 'foo', 'bar', 'baz'}


>>> x1 &= x2
>>> x1
{'foo', 'baz'}

>>> x1.intersection_update(['baz', 'qux'])
>>> x1
{'baz'}
```

SETS

```
>>> x = {'foo', 'bar', 'baz'}

>>> x.discard('baz')
>>> x
{'bar', 'foo'}

>>> x.discard('qux')
>>> x
{'bar', 'foo'}

>>> x.pop()
'bar'
>>> x
{'baz', 'foo'}
```

SETS

Method	Description
add()	Adds an element to the set
difference()	Returns a set containing the difference between two or more sets
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two or more sets
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
symmetric_difference()	Returns a set with the symmetric differences of two sets
union()	Return a set containing the union of sets

DICTIONARIES

```
d = {
    <key>: <value>,
    <key>: <value>,
      .
      .
      .
    <key>: <value>
}
```

```
>>> MLB_team = {
...     'Colorado' : 'Rockies',
...     'Boston'   : 'Red Sox',
...     'Minnesota': 'Twins',
...     'Milwaukee': 'Brewers',
...     'Seattle'  : 'Mariners'
... }

>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',
'Milwaukee': 'Brewers', 'Seattle': 'Mariners'}
```

DICTIONARIES

```
>>> MLB_team['Minnesota']
'Twins'
>>> MLB_team['Colorado']
'Rockies'

>>> MLB_team['Kansas City'] = 'Royals'
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',
'Milwaukee': 'Brewers', 'Seattle': 'Mariners', 'Kansas City': 'Royals'}

>>> MLB_team['Seattle'] = 'Seahawks'
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',
'Milwaukee': 'Brewers', 'Seattle': 'Seahawks', 'Kansas City': 'Royals'}

>>> del MLB_team['Seattle']
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',
'Milwaukee': 'Brewers', 'Kansas City': 'Royals'}
```

DICTIONARIES

```
>>> person = {}
>>> type(person)
<class 'dict'>

>>> person['fname'] = 'Joe'
>>> person['lname'] = 'Fonebone'
>>> person['age'] = 51
>>> person['spouse'] = 'Edna'
>>> person['children'] = ['Ralph', 'Betty', 'Joey']
>>> person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}

>>> person
{'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',
'children': ['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

>>> person['fname']
'Joe'
>>> person['age']
51
>>> person['children']
['Ralph', 'Betty', 'Joey']
```

DICTIONARIES

```
>>> person
{'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',
'children': ['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

>>> person['children'][-1]
'Joey'
>>> person['pets']['cat']
'Sox'
```

DICTIONARIES

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.items())
[('a', 10), ('b', 20), ('c', 30)]
>>> list(d.items())[1][0]
'b'
>>> list(d.items())[1][1]
20
```

DICTIONARIES

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.keys())
['a', 'b', 'c']

>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.values())
[10, 20, 30]
```

DICTIONARIES

```
>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d.pop('b')
20
>>> d
{'a': 10, 'c': 30}

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d.popitem()
('c', 30)
>>> d
{'a': 10, 'b': 20}

>>> d.popitem()
('b', 20)
>>> d
{'a': 10}
```

DICTIONARIES

```
>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d2 = {'b': 200, 'd': 400}

>>> d1.update(d2)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}

>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d1.update(b=200, d=400)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

DICTIONARIES

```python
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

for x in thisdict:
    print(x)
'''brand
model
year'''


for x in thisdict:
  print(thisdict[x])

'''Ford
Mustang
1964'''
```

DICTIONARIES

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

for x in thisdict.keys():
  print(x)
'''brand
model
year'''


for x in thisdict.values():
  print(x)
'''Ford
Mustang
1964'''

for x, y in thisdict.items():
  print(x, y)
'''brand Ford
model Mustang
year 1964'''
```

LIBRARIES

IMPORT LIBRARY

```python
import math

print(math.cos(0))
```

```python
import math as m

print(m.cos(0))
```

```python
from math import cos

print(cos(0))
```

```python
from math import *

print(cos(0))
```

INSTALL LIBRARY

```
pip install numpy
```

```
pip install —upgrade numpy
```

```
pip install numpy==1.23.5
```

```
conda install numpy
```

```
conda install numpy=1.13
```

SYMPY

INSTALL AND IMPORT

```
pip install sympy
```

```
from sympy import *
#or
import sympy as sym
```

SYMBOLS

```
>>>import math
>>>math.sqrt(5) #2.23606797749979
```

```
>>>from sympy import *
>>>sqrt(5)
```

$$\sqrt{5}$$

```
>>>import math
>>>math.sqrt(x) #NameError: name 'x' is not defined
```

```
>>>from sympy import *
>>>x = symbols("x")
>>>sqrt(x)
```

$$\sqrt{x}$$

SUBSTITUTION

```
>>>x, y, z = symbols("x y z")
>>>expr = cos(x) + y
>>>expr
```

$$y + cos(x)$$

```
>>>expr.subs(x, 1)
>>>expr
```

$$y + cos(1)$$

```
>>>expr.subs({x: 1, y: 2})
>>>expr
```

$$cos(1) + 2$$

CONVERT STRING

```
>>>expr = "cos(x) + y**2 + sqrt(z)"
>>>simplify(expr)
```

$$cos(x) + y^2 + \sqrt(z)$$

```
>>>e = simplify(expr)
>>>latex(e)
#y^{2} + \\sqrt{z} + \\cos{\\left(x \\right)}
```

SIMPLIFICIATION

```
>>>expr = (x**3 + x**2 - x -1)/(x**2 + 2*x + 1)
#(x+1)**2 * (x-1)/(x+1)(x-1)
>>>expr
```

$$\frac{x^3 + x^2 - x - 1}{x^2 + 2x - 1}$$

```
>>>factor(expr)
```

$$x - 1$$

SIMPLIFICIATION

```
>>> x = 5*y + cos(z)
>>>expr = (x**3 + x**2 - x -1)/(x**2 + 2*x + 1)
#(x+1)**2 * (x-1)/(x+1)(x-1)
>>>expr
```

$$\frac{-5y + \left(5y + \cos\left(z\right)\right)^{3} + \left(5y + \cos\left(z\right)\right)^{2} - \cos\left(z\right) - 1}{10y + \left(5y + \cos\left(z\right)\right)^{2} + 2\cos\left(z\right) + 1}$$

```
>>>factor(expr)
```

$$5y + \cos\left(z\right) - 1$$

SIMPLIFICIATION

```
>>>expand((x-1) * (x + 1)**2)
```

$$x^3 + x^2 - x - 1$$

```
>>>factor(x** 3 +  x**2 - x - 1)
```

$$\left(x - 1\right)\left(x + 1\right)^2$$

SIMPLIFICIATION

```
>>>cancel((x+1) **3 / (x**2 - 1))
```

$$\frac{x^2 + 2x + 1}{x - 1}$$

```
>>>factor((x+1) **3 / (x**2 - 1))
```

$$\frac{(x + 1)^2}{x - 1}$$

CALCULUS

- Derivatives
- Integrals
- Limits

DERIVATIVES

```
>>>expr = x**2 + sqrt(x)
>>>diff(expr, x)
```

$$2x + \frac{1}{2\sqrt{x}}$$

```
>>>expr = x**2 + sqrt(x)
>>>diff(expr, x, 2)
```

$$2 - \frac{1}{4x^{\frac{3}{2}}}$$

DERIVATIVES

```
>>>expr = x**2 + sqrt(x)
>>>diff(expr, y)
```

$$0$$

```
>>>expr = x**2 + x*sqrt(y)
>>>diff(expr, x, y)
```

$$\frac{1}{2\sqrt{y}}$$

DERIVATIVES

```
>>>expr = x**2 + sqrt(x)
>>>Derivative(expr)
```

$$\frac{d}{dx} \left(\sqrt{x} + x^{2}\right)$$

```
>>>expr = x**2 + x*sqrt(y)
>>>Derivative(expr, y)
```

$$\frac{\partial}{\partial y} \left(x^{2} + x \sqrt{y}\right)$$

DERIVATIVES

```
>>>expr = x**2 + x*sqrt(y)
>>>Derivative(expr, y)
```

$$\frac{\partial}{\partial y}\left(x^{2} + x\sqrt{y}\right)$$

```
>>>expr = x**2 + x*sqrt(y)
>>>Derivative(expr, y, x)
```

$$\frac{\partial^{2}}{\partial x\partial y}\left(x^{2} + x\sqrt{y}\right)$$

DERIVATIVES

```
>>>expr = x**2
>>>expr2 = Derivative(expr, x)
>>>Eq(expr2, expr2.doit())
```

$$\frac{d}{dx} x^{2} = 2x$$

```
>>>expr = x**2 + x*sqrt(y)
>>>expr2 = Derivative(expr, y)
>>>Eq(expr2, expr2.doit())
```

$$\frac{\partial}{\partial y} \left(x^{2} + x\sqrt{y}\right) = \frac{x}{2\sqrt{y}}$$

INTEGRALS

```
>>>expr = x**2
>>>Integral(expr, x)
```

$$\int x^{2}\, dx$$

```
>>>expr = x**2
>>>expr2 = Integral(x**2 + y, x, y)
```

$$\iint \left(x^{2} + y\right)\, dx\, dy$$

INTEGRALS

```
>>>expr = x**2
>>>expr2 = integrate(expr)
```

$$\frac{x^3}{3}$$

```
>>>expr = x**2
>>>expr2 = integrate(expr, (x,0,1))
```

$$\frac{1}{3}$$

INTEGRALS

```
>>>g = Function('g')(x, y)
>>>integrate(g, x, y)
```

$$\iint g(x, y)\, dx\, dy$$

```
>>>g = Function('g')(x, y)
>>>g.integrate(x, y)
```

$$\iint g(x, y)\, dx\, dy$$

INTEGRALS

```
>>>g = Function('g')(x, y)
>>>g.integrate((x, 0, 1), (y, 0, 2))
```

$$\int\limits_0^2 \int\limits_0^1 g(x, y)\, dx\, dy$$

```
>>>g = Function('g')(x, y)
>>>g.integrate((x, 0, 1), (y, 0, x))
```

$$\int\limits_0^x \int\limits_0^1 g(x, y)\, dx\, dy$$

INTEGRALS

```
>>>g = Function('g')(x, y)
>>>integrate(g, (x, 0, 1), (y, x, x**2))
```

$$\int\limits_{x}^{x^{2}}\int\limits_{0}^{1} g{\left(x,y \right)}\, dx\, dy$$

```
>>>g = x**2 + sqrt(y)
>>>integrate(g, (x, 0, 1), (y, 0, x))
```

$$- \frac{2 x^{\frac{3}{2}}}{3} + \frac{2 x^{3}}{3} + \frac{x^{2}}{3} - \frac{x}{3}$$

INTEGRALS

```
>>>g = x**2
>>>integrate(g)
```

$$\frac{x^{3}}{3}$$

```
>>>g = x**2 + sqrt(y)
>>>integrate(g, (x, 0, 1), (y, 0, x))
```

$$- \frac{2x^{\frac{3}{2}}}{3} + \frac{2x^{3}}{3} + \frac{x^{2}}{3} - \frac{x}{3}$$

INTEGRALS

```
>>>g = x**2
>>>integrate(g, (x, 0, 1))
```

$$\frac{1}{3}$$

```
>>>g = x**2
>>>integrate(g, x).doit().subs(x, 3)
```

$$9$$

LIMIT

```
>>>expr = Limit(1/x, x, 0)
>>>Eq(expr, expr.doit())
```

$$\lim_{x\to 0^+} \frac{1}{x} = \infty$$

```
>>>expr = Limit(1/x, x, 2, "+")
>>>Eq(expr, expr.doit())
```

$$\lim_{x\to 2^+} \frac{1}{x} = \frac{1}{2}$$

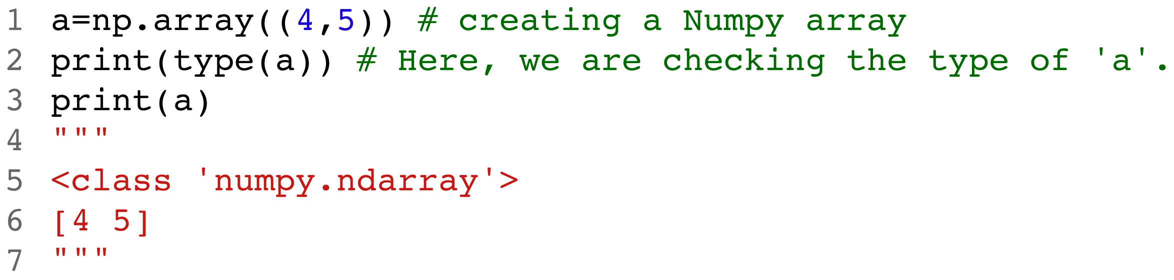
NUMPY

INSTALL AND IMPORT

```
pip install numpy
```

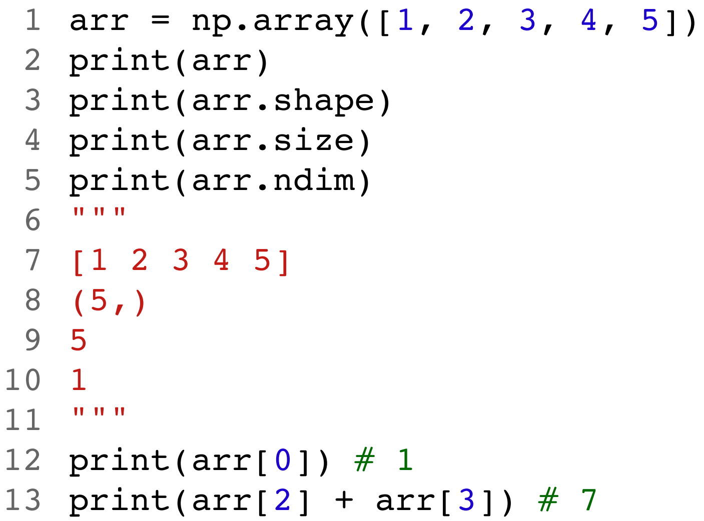
```
from numpy import *
#or
import numpy as np
```

CREATING AN ARRAY



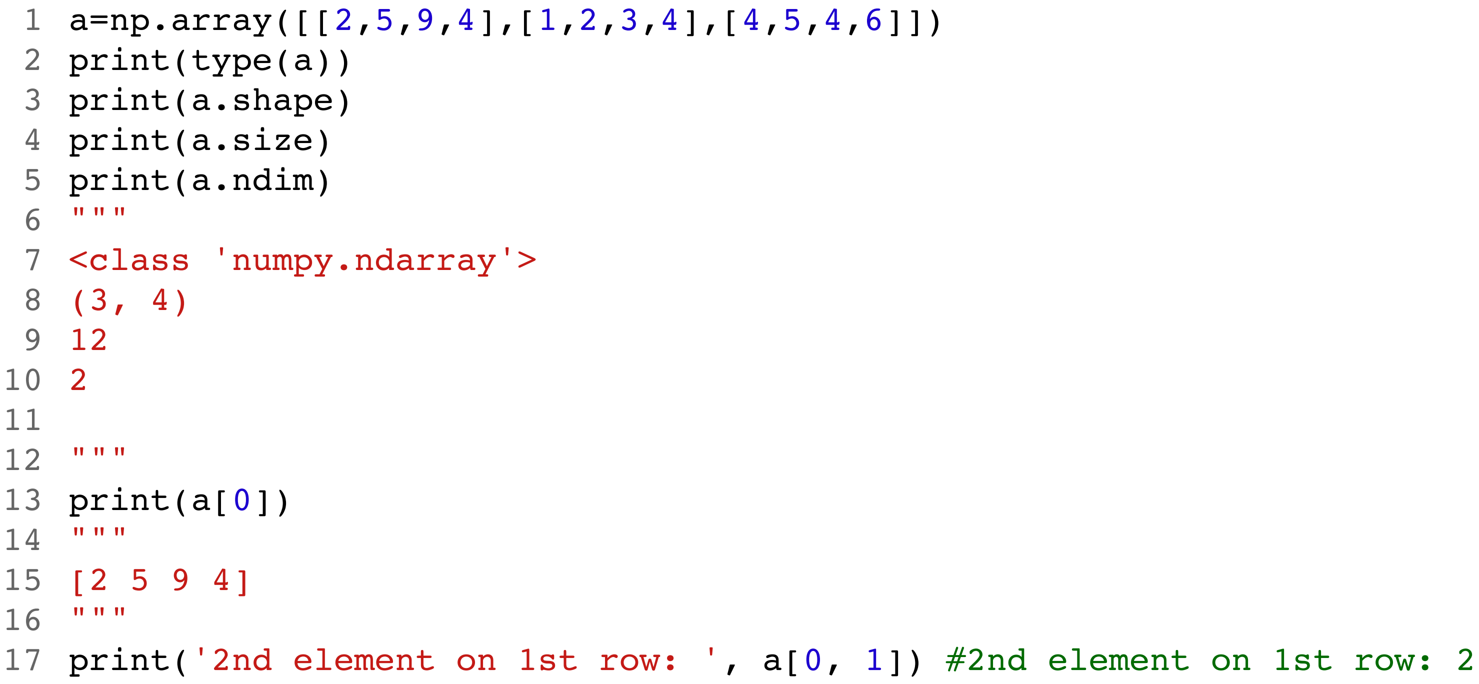
```python
a=np.array((4,5)) # creating a Numpy array
print(type(a)) # Here, we are checking the type of 'a'.
print(a)
"""
<class 'numpy.ndarray'>
[4 5]
"""
```

1-D ARRAY



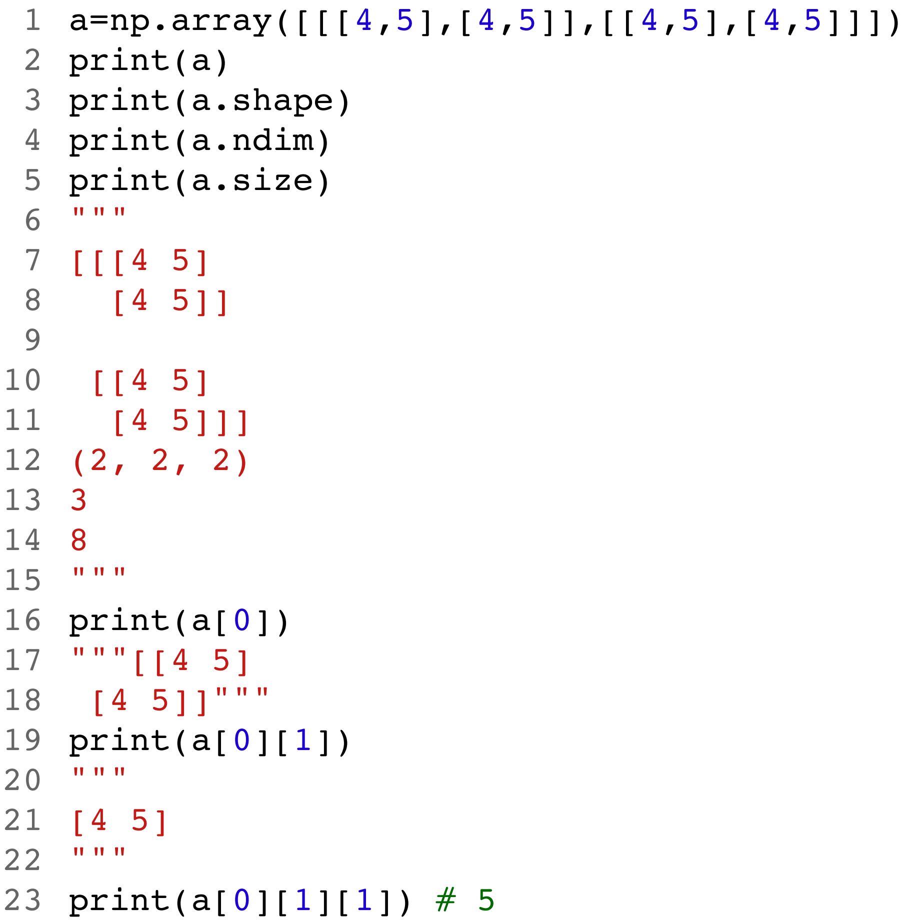
```python
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(arr.shape)
print(arr.size)
print(arr.ndim)
"""
[1 2 3 4 5]
(5,)
5
1
"""
print(arr[0]) # 1
print(arr[2] + arr[3]) # 7
```

2-D ARRAY



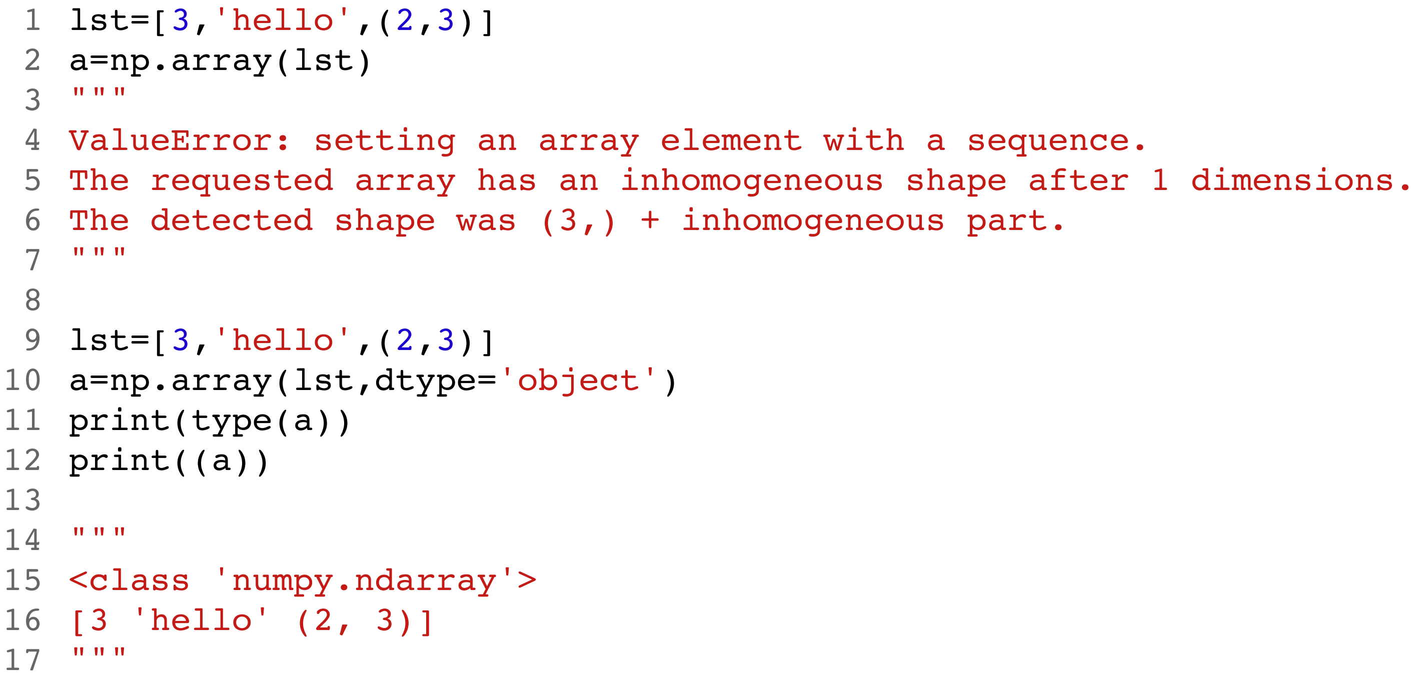
```
a=np.array([[2,5,9,4],[1,2,3,4],[4,5,4,6]])
print(type(a))
print(a.shape)
print(a.size)
print(a.ndim)
"""
<class 'numpy.ndarray'>
(3, 4)
12
2

"""
print(a[0])
"""
[2 5 9 4]
"""
print('2nd element on 1st row: ', a[0, 1]) #2nd element on 1st row: 2
```

3-D ARRAY



```
a=np.array([[[4,5],[4,5]],[[4,5],[4,5]]])
print(a)
print(a.shape)
print(a.ndim)
print(a.size)
"""
[[[4 5]
  [4 5]]

 [[4 5]
  [4 5]]]
(2, 2, 2)
3
8
"""
print(a[0])
"""[[4 5]
 [4 5]]"""
print(a[0][1])
"""
[4 5]
"""
print(a[0][1][1]) # 5
```

OBJECT TYPE



```
lst=[3,'hello',(2,3)]
a=np.array(lst)
"""
ValueError: setting an array element with a sequence.
The requested array has an inhomogeneous shape after 1 dimensions.
The detected shape was (3,) + inhomogeneous part.
"""

lst=[3,'hello',(2,3)]
a=np.array(lst,dtype='object')
print(type(a))
print((a))

"""
<class 'numpy.ndarray'>
[3 'hello' (2, 3)]
"""
```

OBJECT TYPE

```
a=np.array([[4,5],[5]], dtype='object')
print(type(a))
print((a))
print(a.shape)

"""
<class 'numpy.ndarray'>
[list([4, 5]) list([5])]
(2,)
"""
```

INITIALIZED ARRAY

```python
a=np.zeros((4,5),'int32')
print(a,a.dtype)
"""
[[0  0  0  0  0]
[0  0  0  0  0]
[0  0  0  0  0]
[0  0  0  0  0]] int32
"""

a=np.ones((5,5),'int32')
print(a)
"""
[[1  1  1  1  1]
[1  1  1  1  1]
[1  1  1  1  1]
[1  1  1  1  1]
[1  1  1  1  1]]
"""

a=np.empty(2)
print(a)
"""
[ 1.45837449e+105 -5.32504475e-016]
"""
```

INITIALIZED ARRAY

```
a=np.ones((6,2))*6
print(a)
"""
[[6. 6.]
 [6. 6.]
 [6. 6.]
 [6. 6.]
 [6. 6.]
 [6. 6.]]
"""

a=np.full((5,2),6)
print(a)
"""
[[6 6]
[6 6]
[6 6]
[6 6]
[6 6]]
"""
```

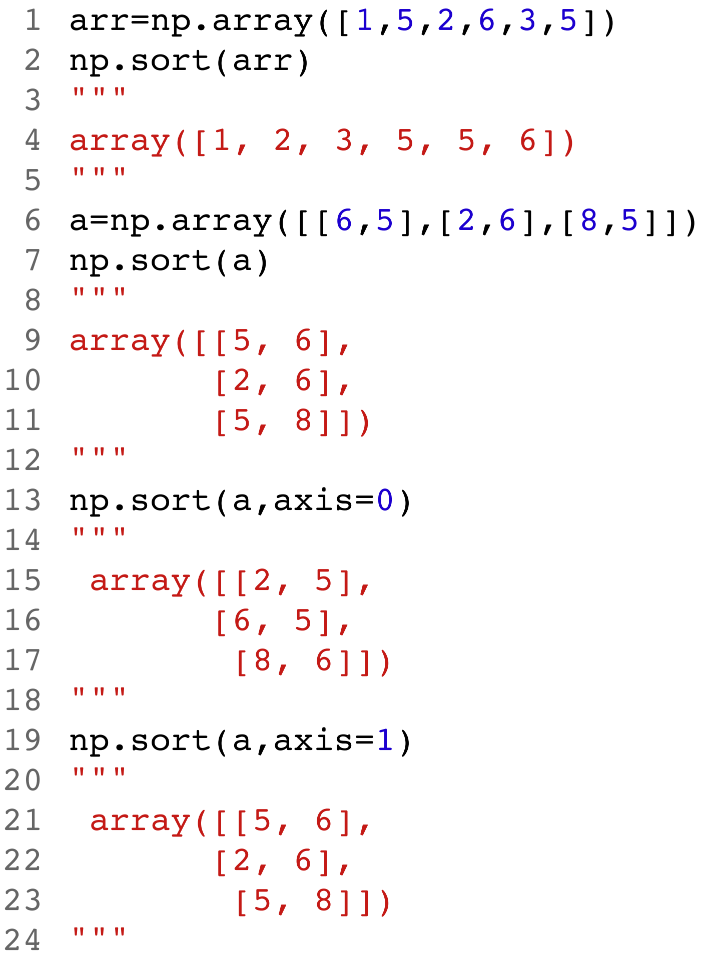
INITIALIZED ARRAY

```python
a=np.eye(5)
print(a)
"""
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
"""

a=np.eye(5,k=2)
print(a)
"""
[[0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
"""
```

INITIALIZED ARRAY

```
e=np.random.random((3,3))
print(e)
"""
[[0.53384421 0.29561298 0.9589127 ]
[0.06149736 0.64260809 0.35274871]
[0.04987664 0.05545438 0.53428942]]
"""

np.linspace(0,10,num=5,dtype='int')
"""
array([ 0, 2, 5, 7, 10])
"""
```

SORT ARRAY



```
arr=np.array([1,5,2,6,3,5])
np.sort(arr)
"""
array([1, 2, 3, 5, 5, 6])
"""
a=np.array([[6,5],[2,6],[8,5]])
np.sort(a)
"""
array([[5, 6],
       [2, 6],
       [5, 8]])
"""
np.sort(a,axis=0)
"""
 array([[2, 5],
       [6, 5],
        [8, 6]])
"""
np.sort(a,axis=1)
"""
 array([[5, 6],
       [2, 6],
        [5, 8]])
"""
```

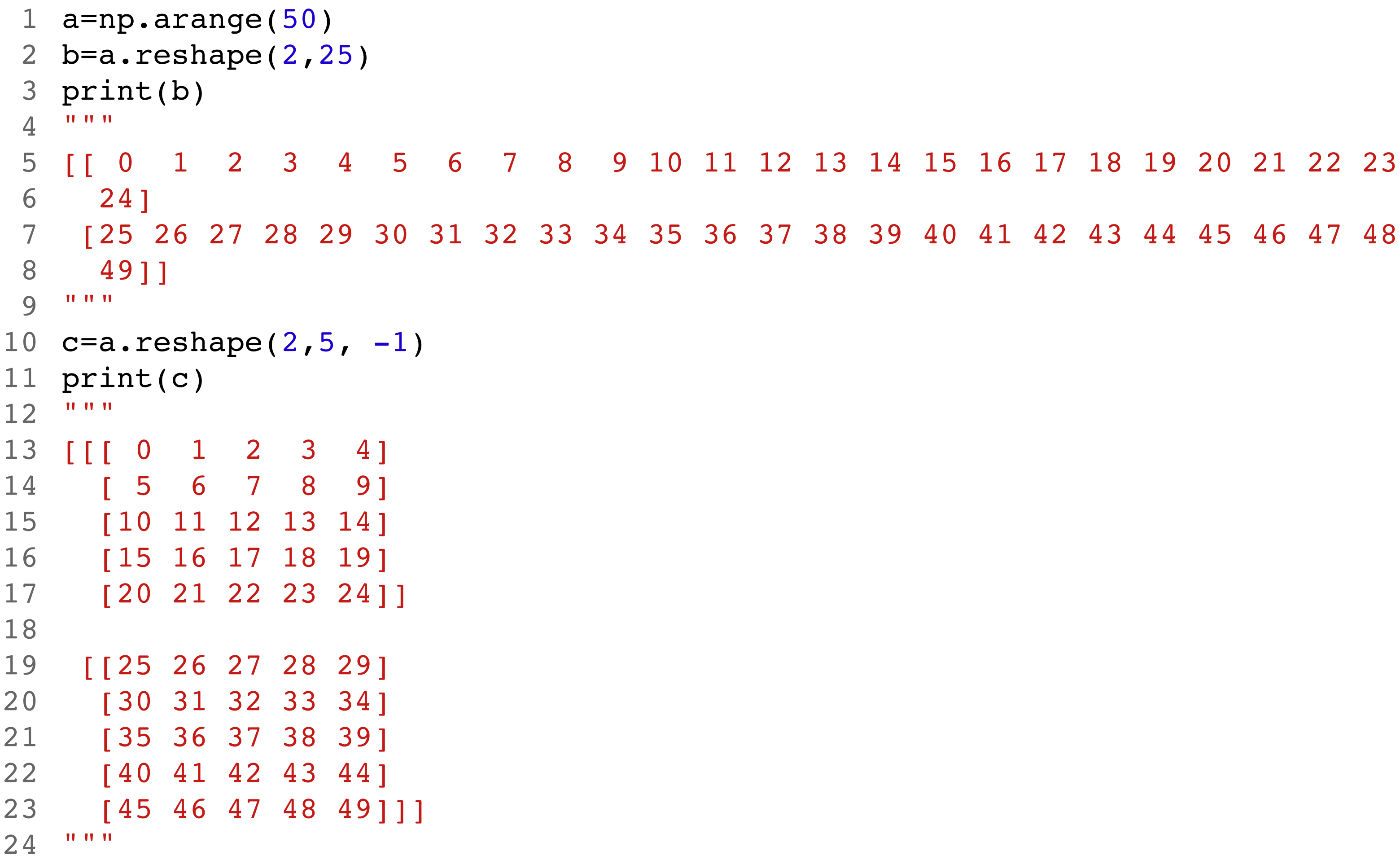
CONCATENATION

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

np.concatenate((a,b))
"""
array([1, 2, 3, 4, 5, 6, 7, 8])
"""
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6]])
np.concatenate((x, y), axis=0)
"""
 array([[1, 2],
       [3, 4],
        [5, 6]])
"""
x=np.array([[1,6],[4,5]])
y=np.array([[5,6],[1,2]])
np.concatenate((x,y),axis=1)
"""
 array([[1, 6, 5, 6],
       [4, 5, 1, 2]])
"""
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6]]) np.concatenate((x, y), axis=None)
"""
array([1, 2, 3, 4, 5, 6])
"""
```

RESHAPE ARRAY

```
a=np.arange(6)
print(a)
b=a.reshape(3,2)
print(b)
"""
[0 1 2 3 4 5]

    [[0 1]
       [2 3]
       [4 5]]
"""
c = np.reshape(a, newshape=(1, 6))
print(c)
"""
[[0, 1, 2, 3, 4, 5]]
"""
```

RESHAPE ARRAY



```
a=np.arange(50)
b=a.reshape(2,25)
print(b)
"""
[[ 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]]
"""
c=a.reshape(2,5, -1)
print(c)
"""
[[[ 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]]]
"""
```

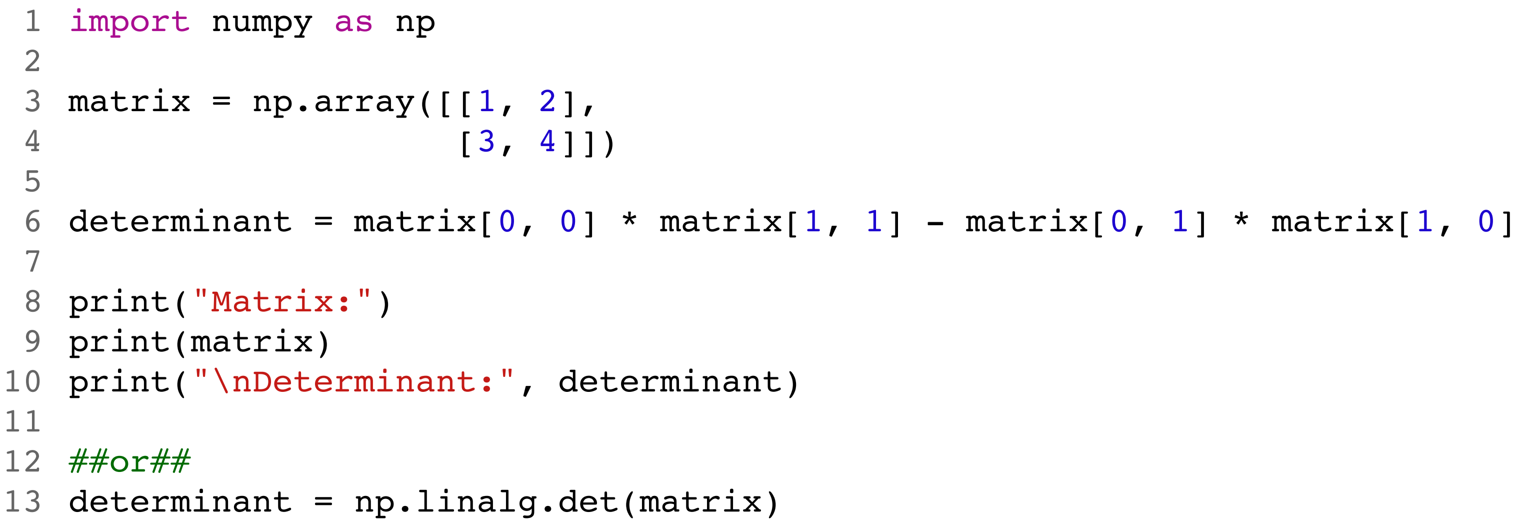
SELECT ELEMENTS

```
 a=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
divisible_by_2=a[a%2==0]
 print(divisible_by_2)
"""
  [2 4 6 8 10 12]
"""
print(a[a < 5])
"""
[1 2 3 4]
"""
c=a[(a>2) & (a<11)]
print(c)
"""
[ 3 4 5 6 7 8 9 10]
"""
c=[(a>2) & (a<11)]
print(c)
"""
[array([[False, False,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True, False, False]])]
"""
```

TRANSPOSE

```python
import numpy as np

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

rows, cols = matrix.shape

transpose_matrix = np.zeros((cols, rows), dtype=matrix.dtype)

for i in range(rows):
    for j in range(cols):
        transpose_matrix[j][i] = matrix[i][j]

print("Original Matrix:")
print(matrix)
print("\nTranspose Matrix:")
print(transpose_matrix)

##or##
transpose_matrix = np.transpose(matrix)
```

DETERMINANT



```python
import numpy as np

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

determinant = matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0]

print("Matrix:")
print(matrix)
print("\nDeterminant:", determinant)

##or##
determinant = np.linalg.det(matrix)
```

DETERMINANT

```python
"""
| a  b  c |
| d  e  f |
| g  h  i |

det(A) = a(ei - fh) - b(di - fg) + c(dh - eg)
"""
import numpy as np

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

# Calculate the determinant manually
determinant = (
    matrix[0, 0] * (matrix[1, 1] * matrix[2, 2] - matrix[1, 2] * matrix[2, 1])
    - matrix[0, 1] * (matrix[1, 0] * matrix[2, 2] - matrix[1, 2] * matrix[2, 0])
    + matrix[0, 2] * (matrix[1, 0] * matrix[2, 1] - matrix[1, 1] * matrix[2, 0])
)

print("Matrix:")
print(matrix)
print("\nDeterminant:", determinant)

##or##
determinant = np.linalg.det(matrix)
```

ADJOINT MATRIX

```python
# Calculate the cofactor matrix
cofactor_matrix = np.array([[(-1)**(i+j) *
                             np.linalg.det(matrix[np.ix_([j, k], [l, m])])
                             for j in range(3)] for i in range(3)])

# Calculate the adjoint matrix (adjugate)
adjoint_matrix = cofactor_matrix.T
```

INVERSE MATRIX

$$A^{-1} = \frac{\operatorname{adj}(A)}{\det(A)}$$

Where:
adj(A) represents the adjoint matrix of matrix (A).
det(A) denotes the determinant of matrix (A).

```
inverse_matrix = adjoint_matrix / determinant

print("Matrix A:")
print(matrix)
print("\nInverse matrix of A:")
print(inverse_matrix)
```

SAVE FILES

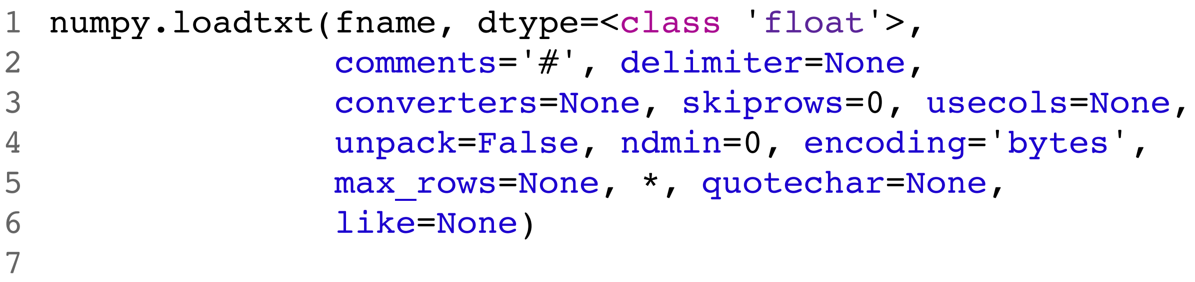
```
numpy.savetxt(fname, X,
              fmt='%.18e', delimiter=' ',
              newline='\n', header='',
              footer='', comments='# ',
              encoding=None)
```

```
random_matrix = np.random.randint(low=0, high=10, size=(4, 5))

print("Random 4x5 matrix: \n", random_matrix)
np.savetxt("file.txt", random_matrix)
"""
Random 4x5 matrix:
[[4 5 3 7 4]
 [5 2 0 1 4]
 [0 1 1 4 3]
 [5 6 4 3 9]]
"""
```

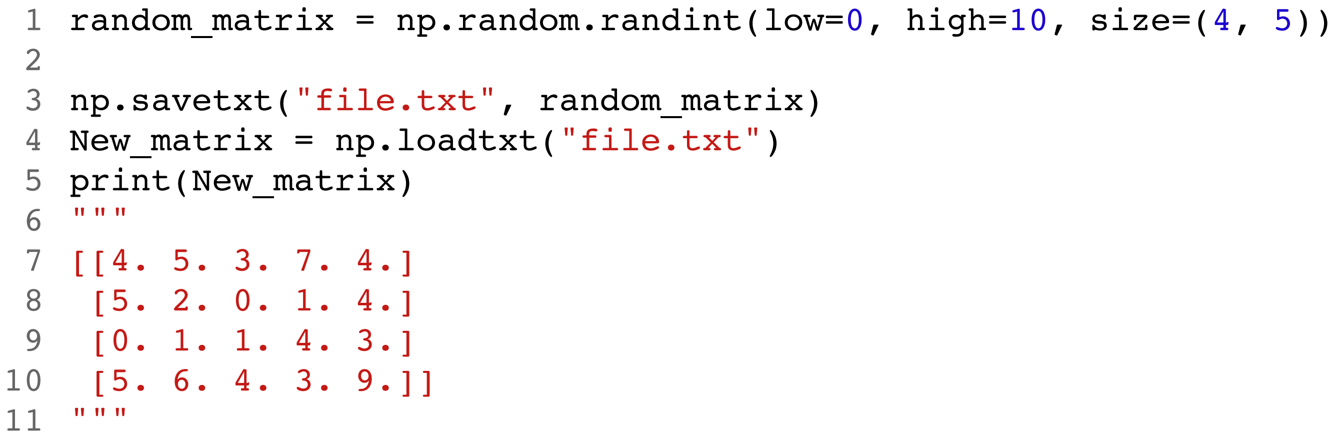
SAVE FILES

Method	Description
fname	Specifies the filename or file handle where the data will be saved.
X	Represents the data (array) that you want to save to the text file.
fmt	A format specifier or sequence of format specifiers for formatting the data.
delimiter	Specifies the string or character separating columns in the output file (default is a space).
newline	Specifies the string or character separating lines in the output file (default is a newline character).
header	A string written at the beginning of the file (optional).
footer	A string written at the end of the file (optional).
comments	A string prepended to the header and footer strings, marking them as comments (optional).
encoding	Specifies the encoding used to encode the output file (optional).

LOAD FILES



```
numpy.loadtxt(fname, dtype=<class 'float'>,
              comments='#', delimiter=None,
              converters=None, skiprows=0, usecols=None,
              unpack=False, ndmin=0, encoding='bytes',
              max_rows=None, *, quotechar=None,
              like=None)

```



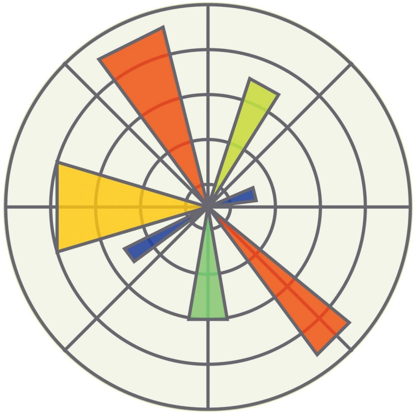
```
random_matrix = np.random.randint(low=0, high=10, size=(4, 5))

np.savetxt("file.txt", random_matrix)
New_matrix = np.loadtxt("file.txt")
print(New_matrix)
"""
[[4. 5. 3. 7. 4.]
 [5. 2. 0. 1. 4.]
 [0. 1. 1. 4. 3.]
 [5. 6. 4. 3. 9.]]
"""
```

LOAD FILES

Method	Description
fname	Specifies the filename or file handle from which to read the data.
dtype	Data type of the resulting array (default is `float`). If a structured data type is used, each row is interpreted as an element of the array.
comments	Characters or list of characters indicating the start of a comment (default is `'#'`). None implies no comments.
delimiter	Character used to separate values (default is whitespace).
converters	Converter functions to customize value parsing (optional).
skiprows	Number of lines to skip at the beginning (including comments, default is 0).
usecols	Which columns to read (default is all columns).
unpack	If True, the returned array is transposed (optional, default is False).

LOAD FILES

Method	Description
ndim	Specifies the minimum number of dimensions of the resulting array (default is 0).
encoding	Specifies the encoding used to encode the input file (default is `'bytes'`).
max_rows	Limits the number of rows read from the file (new in version 1.17.0).
quotechar	Specifies the character used to quote fields (new in version 1.17.0).
like	An existing array-like object that provides the data type and delimiter (new in version 1.20.0).



MATPLOTLIB

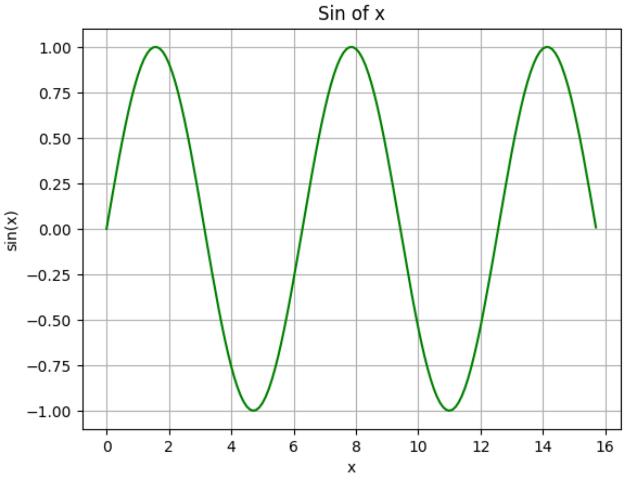
INSTALL AND IMPORT

```
pip install matplotlib
```

```python
from matplotlib import *
#or
import matplotlib as mpl
```

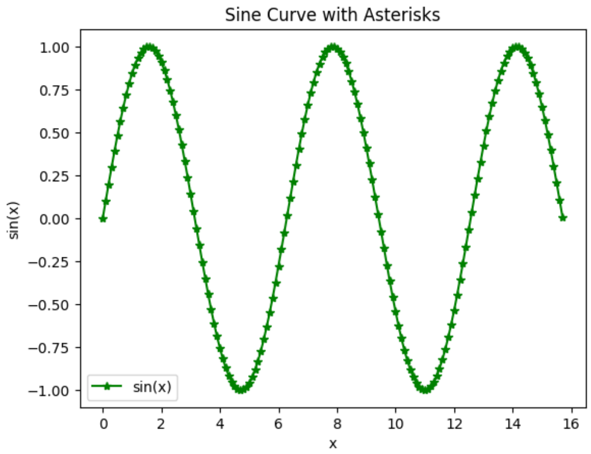
PLOT

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

x = np.arange(0, 5*np.pi, 0.1)
y = np.sin(x)

# Plot the sine curve
plt.plot(x, y, color='green')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('Sin of x')
plt.grid(True)
plt.show()
```



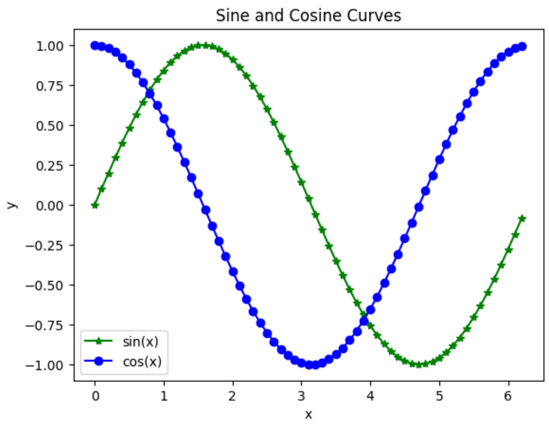
PLOT

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

x = np.arange(0, 5*np.pi, 0.1)
y = np.sin(x)

# Plot the sine curve with asterisks as markers
plt.plot(x, y, color='green', marker='*', linestyle='-',
         markersize=6, label='sin(x)')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('Sine Curve with Asterisks')
plt.grid(True)
plt.legend()
plt.show()
```



PLOT

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

x = np.arange(0, 2*np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the sine curve with asterisks as markers
plt.plot(x, y_sin, color='green', marker='*', linestyle='-',
         markersize=6, label='sin(x)')

# Plot the cosine curve with circles as markers
plt.plot(x, y_cos, color='blue', marker='o', linestyle='-',
         markersize=6, label='cos(x)')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Sine and Cosine Curves')
plt.legend()
plt.show()
```


Sine and Cosine Curves
y
x
sin(x)
cos(x)