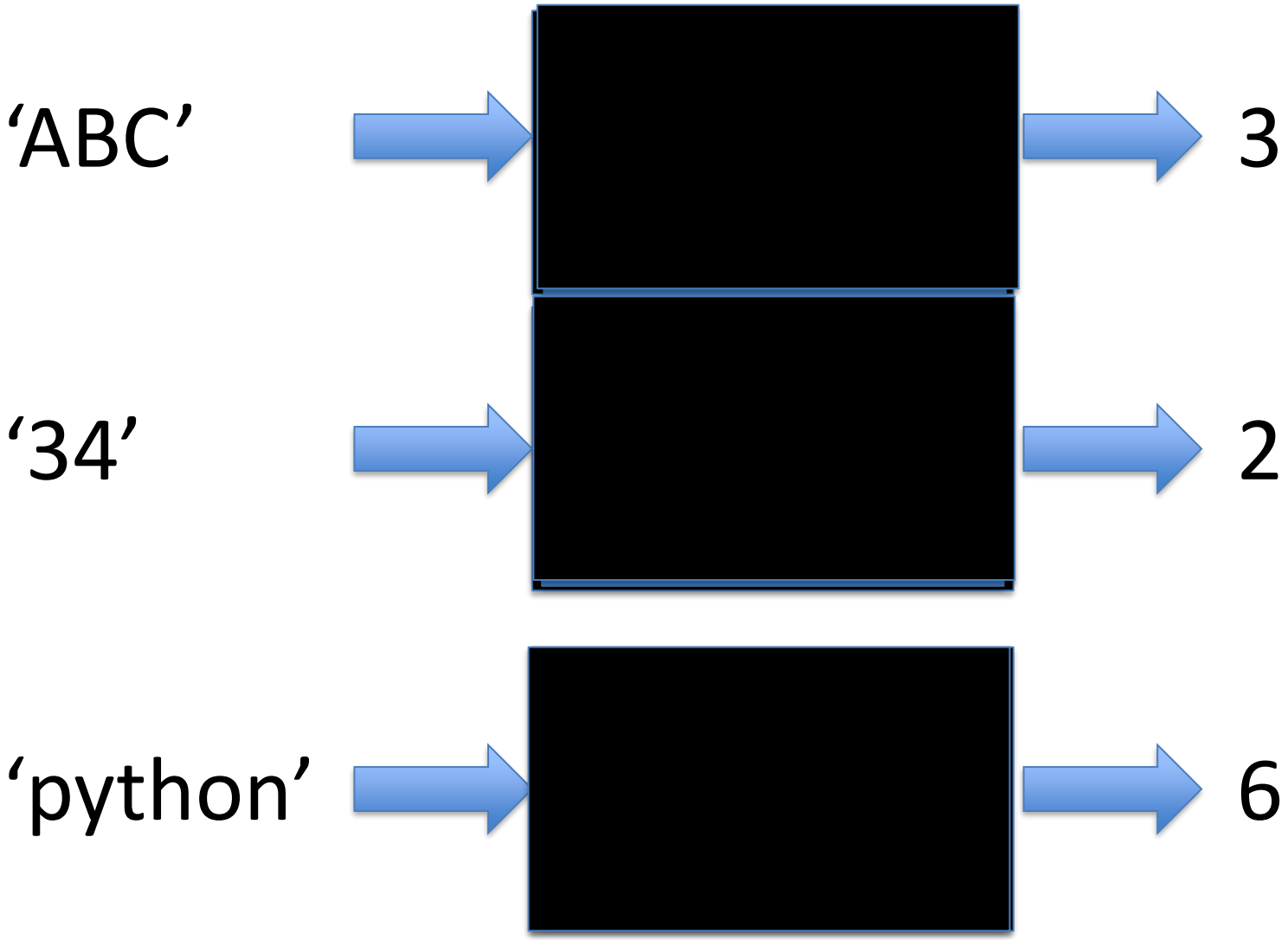
Course	
Term	
Week	
Date	
Chapter. Topic	5. Functions

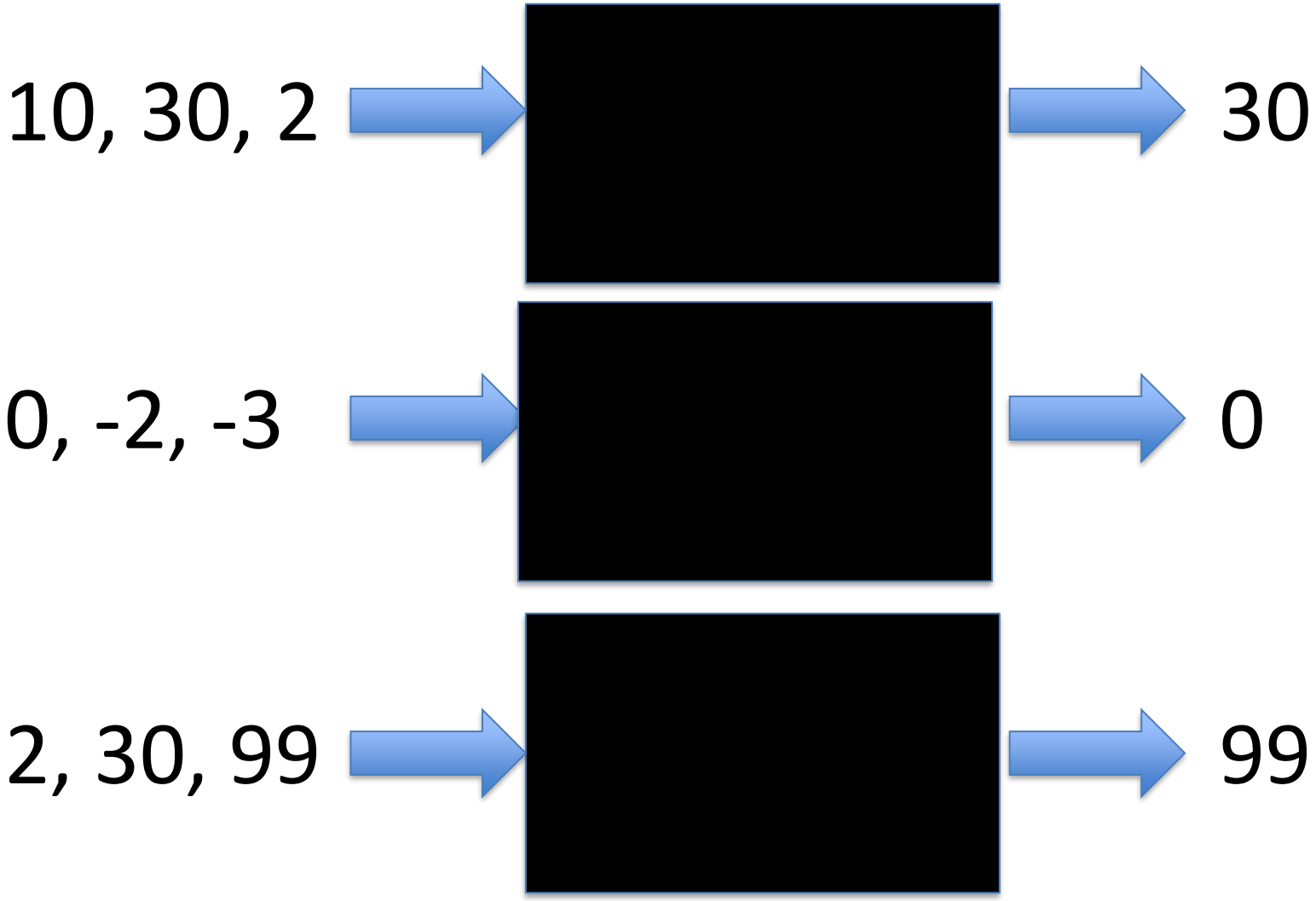
Functions

Siva R Jasthi
Siva.Jasthi@normandale.edu

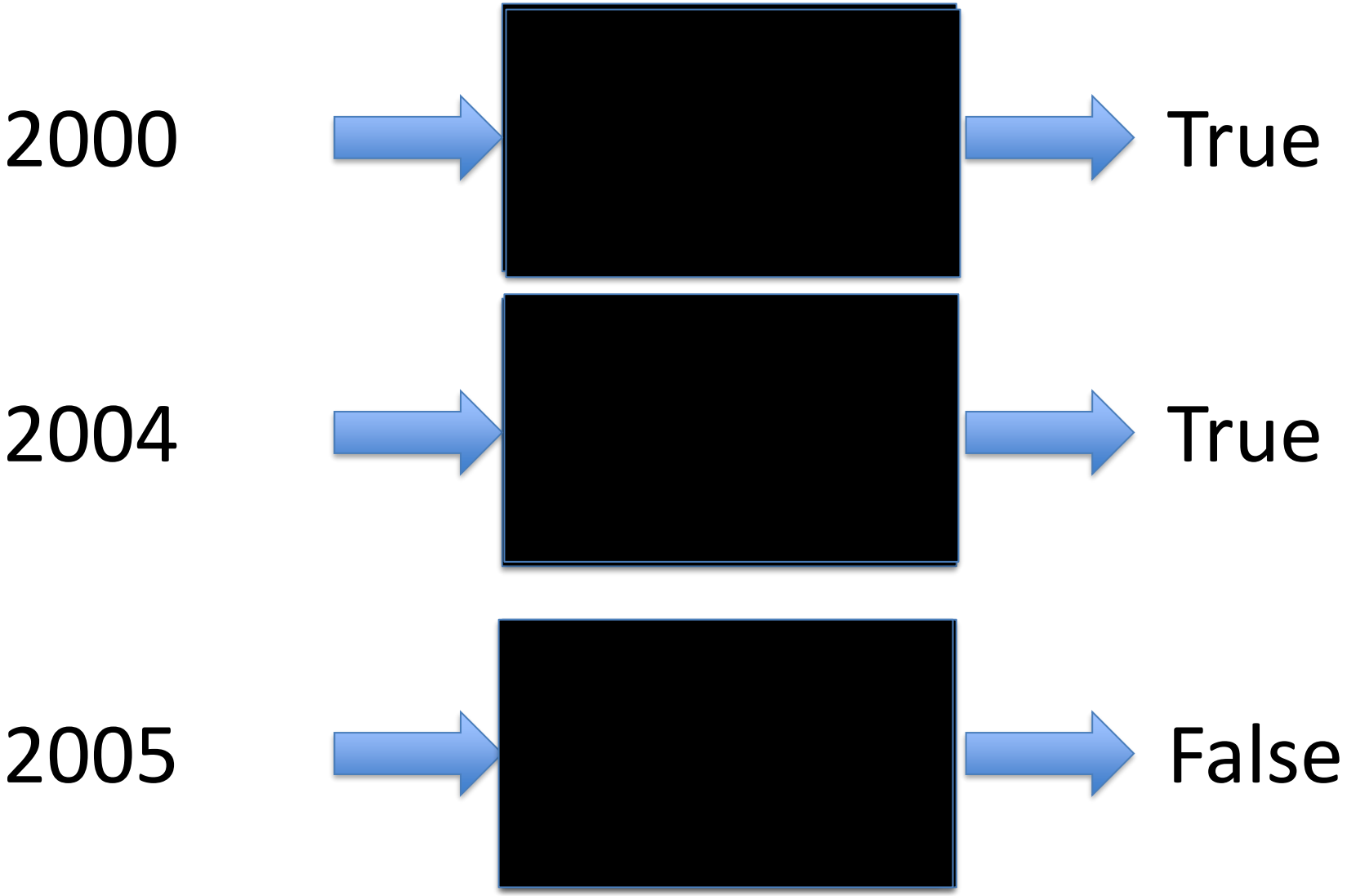
Black Box: Inputs, Processing, Outputs (IPO)



Black Box: Inputs, Processing, Outputs (IPO)

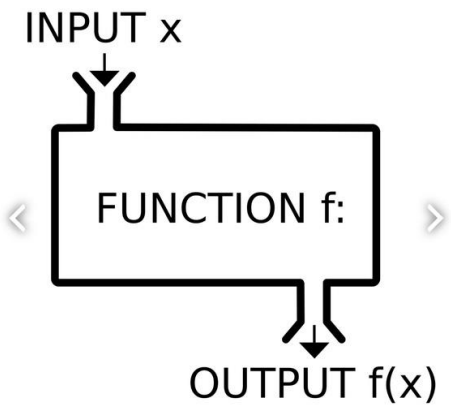


Black Box: Inputs, Processing, Outputs



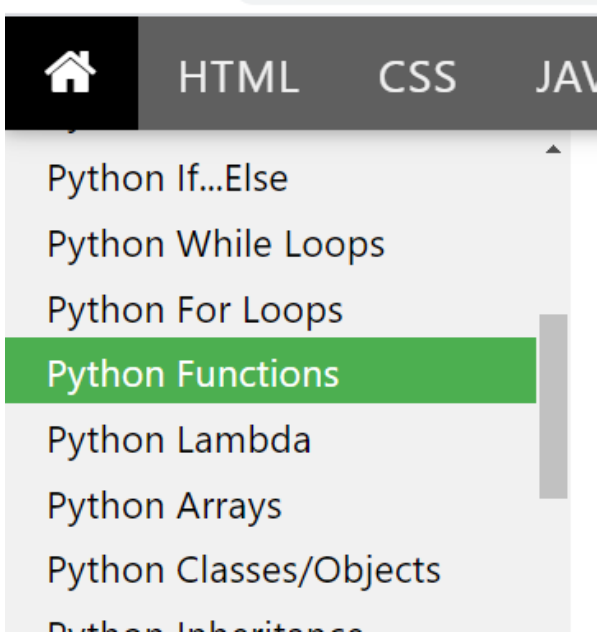
Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.



Review and practice “Try It” exercises here

https://www.w3schools.com/python/python_functions.asp



Functions end with round brackets (parenthesis)

- input(), print(), type( )
- int( ), float( ), str( ), bool( )
- randint(), range()
- len() max() min() sum()

- interst_rate
- speed
- budget
- total
- count

Inputs, Processing, Outputs (IPO)

- Inputs: 0, 1 or N
- Processing: black box
- Outputs: 0, 1 or N

- **Summary**:
  - Any function can take any number of inputs (including 0 inputs)
  - And return any number of outputs (including 0 returns)

Input, Processing, Output (IPO): Chart

Function Name	Inputs (I)	Processing (P)	Output (O)
getMax( )	3 numbers	Find the maximum of three numbers given	1 number
getLength( )	1 String	Find the length of the string	1 number
isLeapYear( )	1 number (year)	Find out whether the given year is leap year or not	True or False
printSequence( )	1 number	Print the number sequence	None
getInitials()	1 String (full nam)	This function returns the first char of first name, middle name and last name	3 chars

Function = Do, Act, Service, Operation, Action, Process, Procedure, Doing something,

Best Practice: Try to name your functions reflecting some “action” or “verb”

We can define our own functions

Area of a Square

f(x) = x*x

Area of a Circle

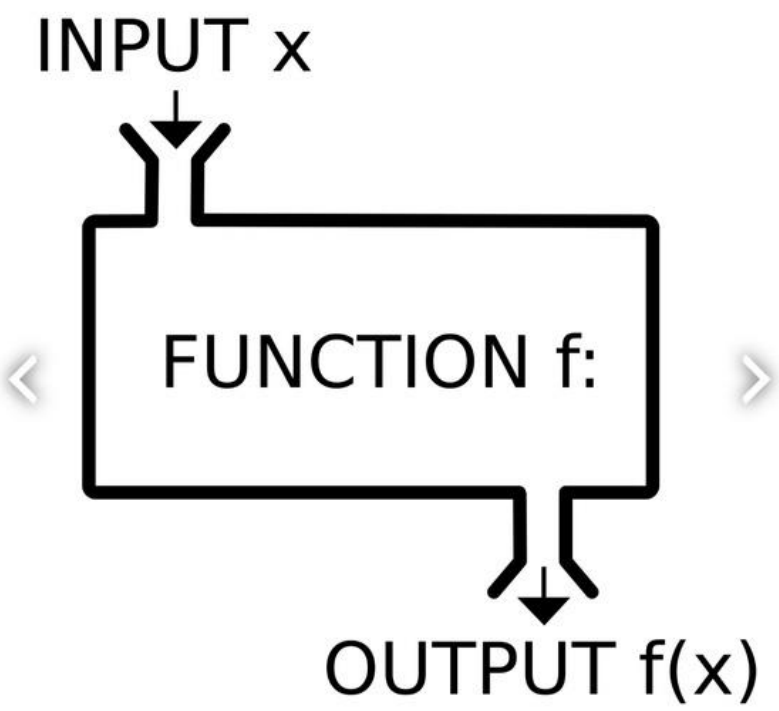
f(y) = (22/7) * y * y

Area of a Rectangle

f(a,b) = a * b

Perimeter of a Rectangle

f(a,b) = (2*a) + (2*b)



def keyword

We use def keyword for defining the functions in Python.

Let us explore some code snippet at this link.

```python
# ======= defining the functions

def area_of_square(x):
    return x*x

def area_of_circle(x):
    import math
    area = (math.pi) * x * x
    return area

def area_of_rectangle(x,y):
    area = x * y
    return area

def perimeter_of_rectangle(x,y):
    perimeter = (2*x) + (2*y)
    return perimeter
```

```python
# === invoking / calling the functions

x = area_of_square(10)
print(x)

y = 10
z = area_of_circle(y)

a = 5
b = 10
c = area_of_rectangle(a,b)
d = perimeter_of_rectangle(a,b)
```

Pattern

Functions are defined **once**.

Functions are called / invoked **many** times.

```python
# ======= defining the functions

def area_of_square(x):
    return x*x


# === invoking / calling the functions

# One function can be called any number of times
# test case 1
p = 10
q = area_of_square(p)
print(q)

# test case 2
x = area_of_square(10)
print(x)

# test case 3
print(area_of_square(5))

```

Formal vs Actual Parameters

At the definition time,

- the parameters are "**formal**"
- a and b are **formal** parameters

```
def max(a,b):
    if a>b:
        return a
    else:
        return b
```

At the calling/invoking time,

- The parameters are "**actual**"
- "actual" parameters can be literals or variables or expressions.
- The number and order should be same.

```
x=10
y=20
z = max(x,y)
print(z)

p = max(4848, 989)
print(p)

print(max(890,45))
```

Parameters vs Arguments vs Args

In programming literature, these are interchangeably used while discussing the functions.

All these mean same thing!

- Parameters
- Arguments
- Args

- Defining Functions (“formal” parameters or “formal” arguments)
- Calling Functions (“actual” parameters or “actual” arguments)

Only thing that matters is: the name, number and type of parameters must match for the correct function to be invoked.

“formal” parameters = arguments
“actual” parameters = parameters

What happens if the parameters do not match?

When calling functions, the number and order should match.

What happens if they do not match?

Let us explore!

```
8  c = max(10)
9  print(c)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (2 steps)

TypeError: max() missing 1 required positional argument: 'b'

```python
def max(a,b):
    if a>b:
        return a
    else:
        return b


c = max(10)
print(c)
```

What happens if the parameters do not match? (contd.)

When calling functions, the number and order should match.

What happens if they do not match?

Let us explore!

```
7
8  a = max(10,30, 15)
9  print(a)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (2 steps)

TypeError: max() takes 2 positional arguments but 3 were given

```
1  def max(a,b):
2      if a>b:
3          return a
4      else:
5          return b
6
7
8  a = max(10,30, 15)
9  print(a)
```

Types of Functions

Value-Returning Functions (example)

Void-Returning Functions (they return "None") (example)

```python
# Given a name, return the initials of first and last names
def getInitials(input_str):
    tokens = input_str.split()
    first_initial = tokens[0][0]
    last_initial = tokens[-1][0]
    total_initials = first_initial + last_initial
    return total_initials


# Testing
x = getInitials('Mona Lisa')
print(x)

y = getInitials('Siva Rama Krishna Jasthi')
print(y)
```

```python
# Given a name, this method greets the user
# this is an example 'void returning" function
def greet(name):
    print('Hello ', name)
    print('Welcome to Python Programming!')


# Testing
x = greet('Siva Jasthi') # x is None
print(x)

# There is no value in assigning the function to any variable
# because it is nothing or NULL or void
greet('John Doe')
```

“void” functions

Did you notice the difference between “serve” function and “max” function.

- Our “max” function returns a value.
- However, “serve” function does not return any value.

```
def serve(sandwich_type, drink_size, fries_size):
    print("Here is your order: ")
    print(sandwich_type)
    print(drink_size)
    print(fries_size)
    print()
```

“void” function
No return value

Functions that do not return any value are called **“void”** functions. They do some useful stuff (repeatable instructions) but caller will not be given any value back.

```
def max(a,b):
    if a>b:
        return a
    else:
        return b
```

Function with
“return” value

How many inputs? And returns?

Inputs can be 0, 1 or more.

Returns can be 0, 1 or more

Type	Inputs	Returns	Example
A	0	0	method_A( )
B	0	1	method_B( )
C	0	N	method_C( )
D	1	0	method_D( )
E	1	1	
F	1	N	
G	N	0	
H	N	1	
I	N	N	

```python
# 0 inputs, 0 returns
def method_A():
    print('welcome to python programming!')
    print('All our classes are held on ZOOM')
    print('All material is in google classroom')


# call the methods
method_A()
```

```python
# 0 inputs, 1 return
# give me a random vowel
def method_B():
    import random
    vowels = 'aeiou'
    random_vowel = random.choice(vowels)
    return random_vowel


# call the methods
r_vowel = method_B()
print('random vowel: ', r_vowel)
```

```python
# 0 inputs, N returns
# give me two consonants
def method_C():
    import random
    consonants = 'bcdfghjklmnpqrstvwxyz'
    random_c1 = random.choice(consonants)
    random_c2 = random.choice(consonants)
    while (random_c1 == random_c2):
        random_c2 = random.choice(consonants)
    return random_c1, random_c2


# call the methods
r_consonants = method_C()
print('random consonants: ', r_consonants)
```

```python
# 1 input, 0 returns
# greet the user
def method_D(name):
   print('Welcome', name)
   print('Have fun programming')


# call the methods
method_D('Jane Doe')
```

```
def withdraw(amount):
    print(f"You are withdrawing ${amout} from your account")
    return amount

amount_needed = withdraw(100)
print("I now got ", amount_needed)   # I now got 100
```

==========

```
def withdraw(amount):
    print(f"You are withdrawing ${amout} from your account")

amount_needed = withdraw(100)
print("I now got ", amount_needed)    # I now got None
```

PYTHON PROGRAMMING by SIVA JASTHI

```python
#@title 1 input, 1 returns
# convert a decimal to a binary
def method_E(number):
    binary_number = bin(number)
    print(binary_number)
    return binary_number.replace("0b", "")


# call the methods
b_10 = method_E(10)
print("Binary of 10 : ", b_10)
```

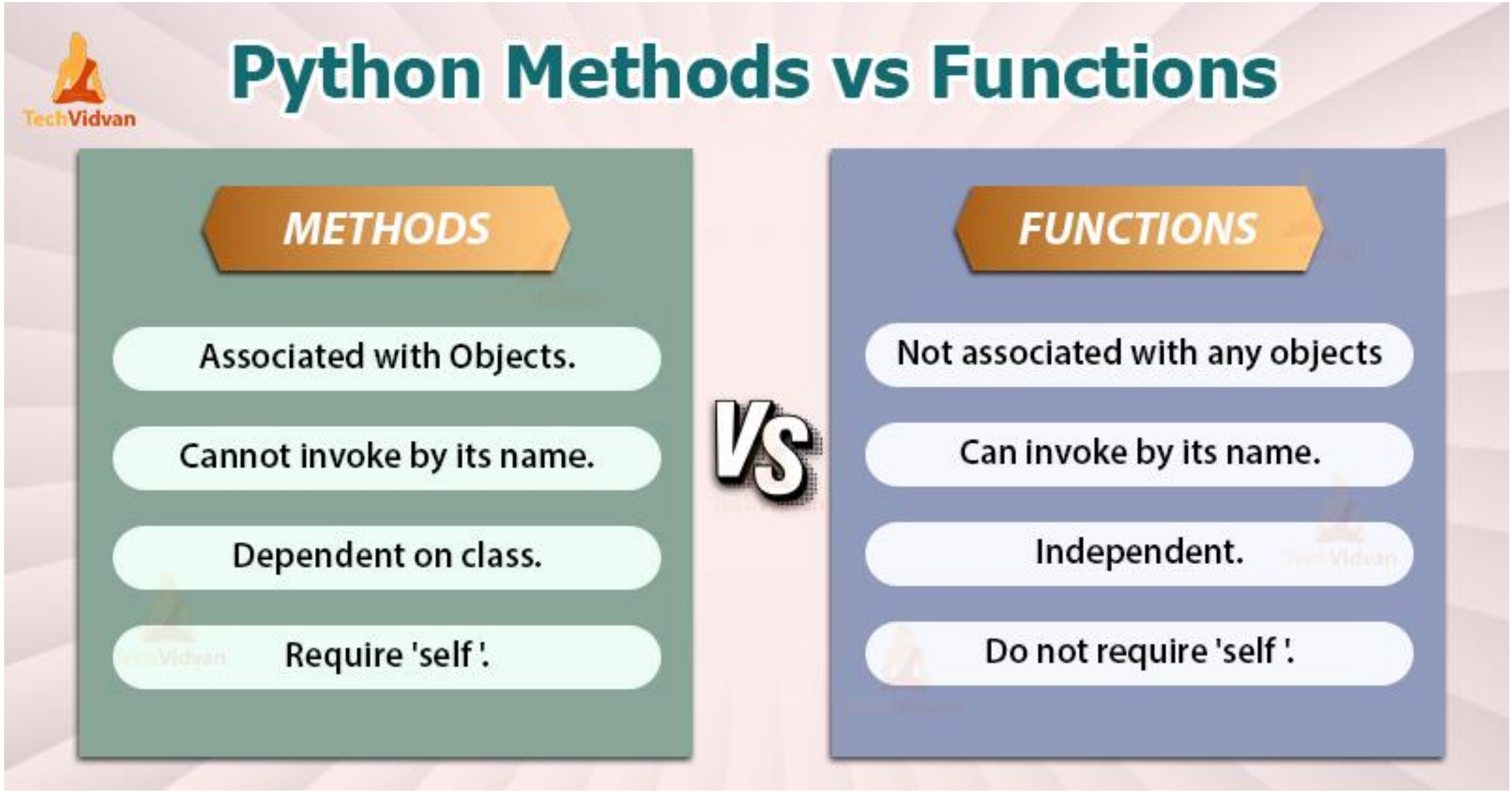
```python
#@title N inputs, 0 returns
# Given first name, middle name and last name, print the initials
def method_G(f_name, m_name, l_name):
  print(f_name[0], m_name[0], l_name[0])
  print(f_name[0] + m_name[0] + l_name[0])

# invoke the method
method_G('siva', 'rama', 'jasthi')
```

```python
#@title 1 input, N returns
# Given a string, return the count of vowels and consonants
def method_F(input_str):
   vowels = 'aeiou'
   v_count = 0
   c_count = 0
   for elem in input_str:
     if elem in vowels:
       v_count = v_count + 1
     else:
       c_count = c_count + 1

   return v_count, c_count


# call the methods
str_1 = 'Python'
x = method_F(str_1)
print("Vowel count: ", x[0], " Consonant Count: ", x[1]);

str_1 = 'Python'
v, c = method_F(str_1)
print("Vowel count: ", v, " Consonant Count: ", c);
```

```python
#@title N inputs, 1 returns
# Given sales price and sales tax, find the total cost
def method_H(price, tax_rate):
  total_price = price + (price * tax_rate / 100)
  return total_price

# invoke the method
t_p = method_H(200, 5)
print('Total Price: ', t_p)

t_p = method_H(400, 10)
print('Total Price: ', t_p)
```

```python
#@title N inputs, N returns
# Give a list of numbers, find its max, min, range, length, mean
def method_I(numbers):
  max_value = max(numbers)
  min_value = min(numbers)
  range = max_value - min_value
  length = len(numbers)
  mean = sum(numbers) / length
  return max_value, min_value, range, length, mean

# invoke the method
number_list = [1, 2, 3, 4, 5]
stats = method_I(number_list)
print('Max: ', stats[0])
print('Min: ', stats[1])
print('Range: ', stats[2])
print('Length: ', stats[3])
print('Mean: ', stats[4])
```

Functions vs Methods



Difference between function vs method

Functions vs. methods

A **method is a specific kind of function** - it behaves like a function and looks like a function, but differs in the way in which it acts, and in its invocation style.

A **function doesn't belong to any data** - it gets data, it may create new data and it (generally) produces a result.

A method does all these things, but is also able to **change the state of a selected entity**.

A method is owned by the data it works for, while a function is owned by the whole code.

This also means that invoking a method requires some specification of the data from which the method is invoked.

It may sound puzzling here, but we'll deal with it in depth when we delve into object-oriented programming.

In general, a typical function invocation may look like this:

```
result = function(arg)
```

The function takes an argument, does something, and returns a result.

Python’s built-in functions (not owned by anyone)

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

https://www.w3schools.com/python/python_ref_functions.asp

Difference between function vs method (Contd.)

A typical method invocation usually looks like this:

```
result = data.method(arg)
```

Note: the name of the method is preceded by the name of the data which owns the method. Next, you add a **dot**, followed by the **method name**, and a pair of **parenthesis enclosing the arguments**.

The method will behave like a function, but can do something more - it can **change the internal state of the data** from which it has been invoked.

Functions: Summary

Functions:

- Promote functional programming
- Support the code reuse
- Improves the program readability
- Once proven (tested), we can take those for granted when we use them.

A group of related functions are kept in a library called “module”.
For example, “string”, “random”, “math” – are modules. We import those modules and use those functions as needed.

Thank You.