PYTHON PROGRAMMING by SIVA JASTHI

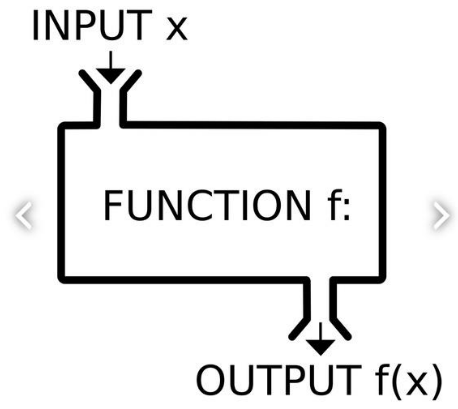
Course	
Term	
Week	
Date	
Chapter. Topic	5. Functions

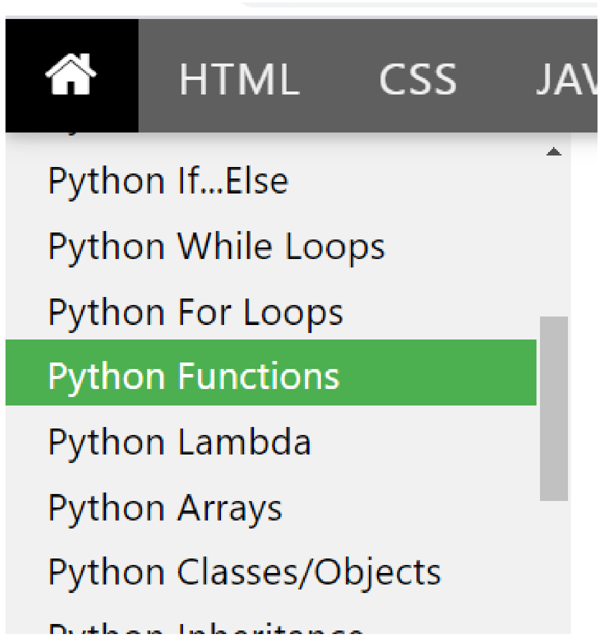
Positional and Keyword Arguments
Variable Number of Arguments (*args, **kwargs)
Yield (aka Generator Functions)

Siva R Jasthi
Computer Science and Cybersecurity
Metropolitan State University

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

What did we learn so far?

- Functions are building blocks
- Three things matter:
  - Fuction Name,
  - Number of Arguments and
  - Position of Arguments
- Void-returing vs Value-Returning Functions
- Functions can take 0, 1, or N arguments as inputs
- Functions can return 0, 1, or N arguments to the callers

Outline for today

- Positional Arguments (number and order – these two matter)
- Keyword Arguments (aka “named parameters”)
- Optional Arguments (aka Default Values)

Advanced Topics:

- Variable Number of Arguments (*args)
- Variable Number of Keyword Arguments (**kwargs)

Positional and Keyword Arguments

- Python functions can contain two types of arguments:
  - *positional arguments* and
  - *keyword arguments*.

- Positional arguments must be included in the correct order.

- Keyword arguments are included with a keyword and equals sign.

Postional Arguments

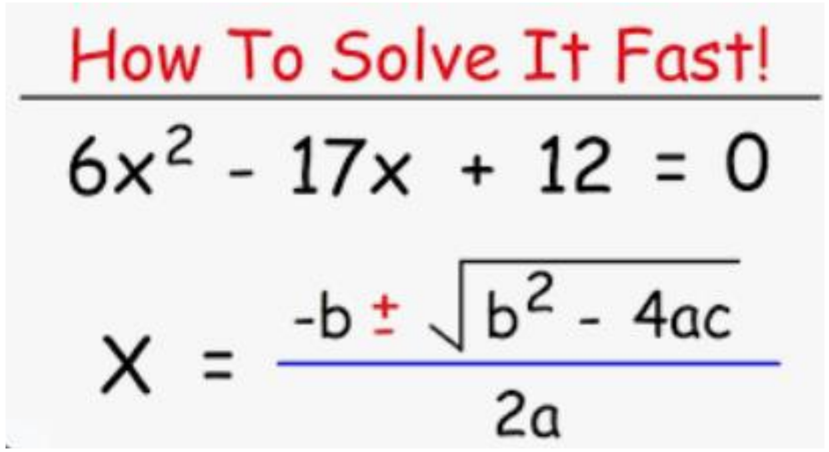
PYTHON PROGRAMMING by SIVA JASTHI

Positional Arguments

# quadratic equation a*x*x + b*x + c = 0
- def quadratic_equation(a, b, c):

# Calling the above function (correct way)
quadratic_equation(6, -17, 12)

# Calling the above function (wrong way)
quadratic_equation(12, 6, -17)


How To Solve It Fast!

$$6x^2 - 17x + 12 = 0$$

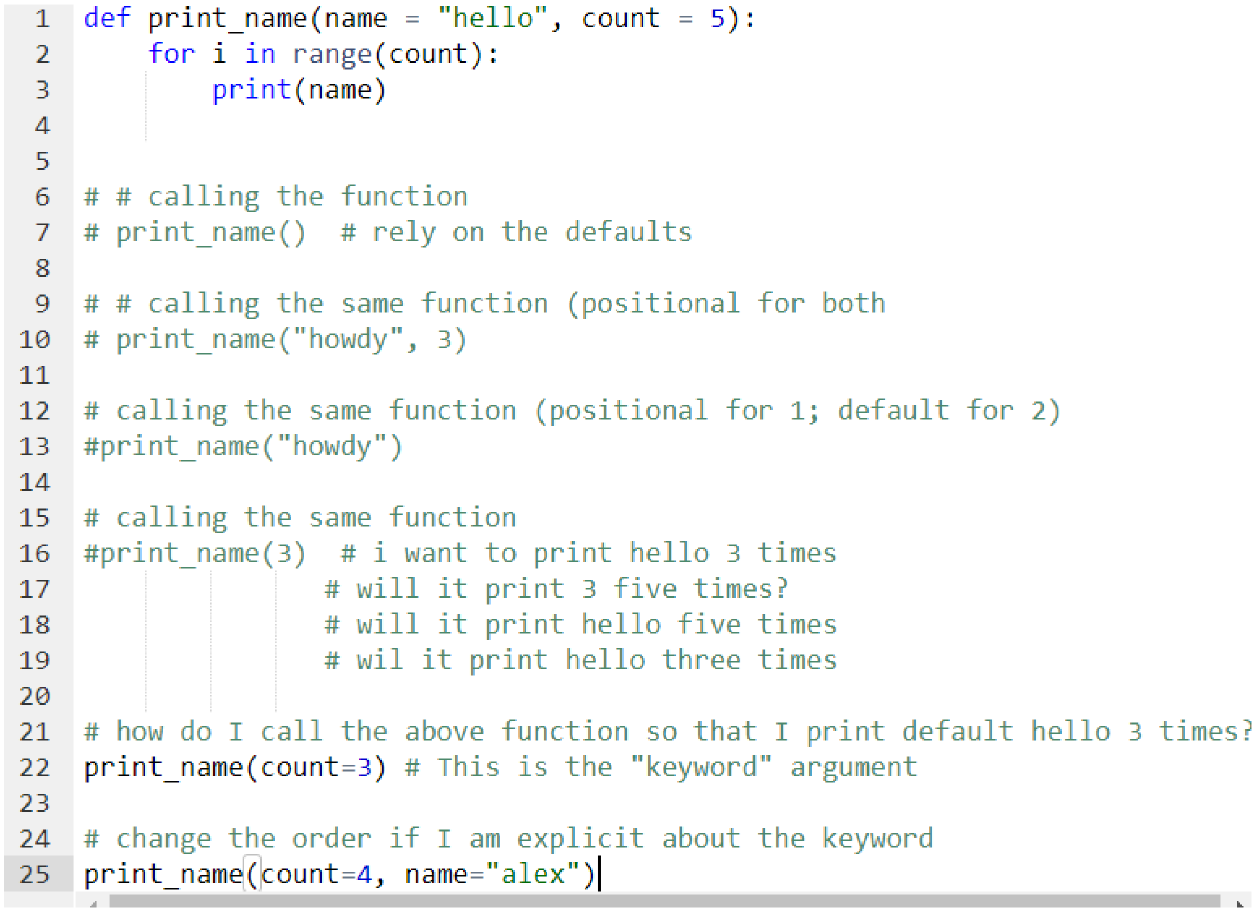
$$X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Why is it wrong?
The “position” of an argument matters.
First argument should be passed in first.
Second argument should appear next and so on.

Keyword Arguments

Keyword Arguments (pythontutor)

- Callers can invoke the functions with explicit mapping of the variables and their values.
- When they do, the position doesn't matter.
- Because of the explicit name you are giving, python binds the values.
- For all advanced data science libraries, this is typically the way the methods are invoked.



```python
def print_name(name = "hello", count = 5):
    for i in range(count):
        print(name)


# # calling the function
# print_name()  # rely on the defaults

# # calling the same function (positional for both
# print_name("howdy", 3)

# calling the same function (positional for 1; default for 2)
#print_name("howdy")

# calling the same function
#print_name(3)  # i want to print hello 3 times
               # will it print 3 five times?
               # will it print hello five times
               # wil it print hello three times

# how do I call the above function so that I print default hello 3 times?
print_name(count=3) # This is the "keyword" argument

# change the order if I am explicit about the keyword
print_name(count=4, name="alex")
```

Keyword Arguments (pythontutor)

- How can I force my callers (consumers, or customers, or end-users) to explicitly specify the argument name when they call me?

- (NO) print_name(“hi”, 10)

- (YES) print_name(name=“hi”, count=10)

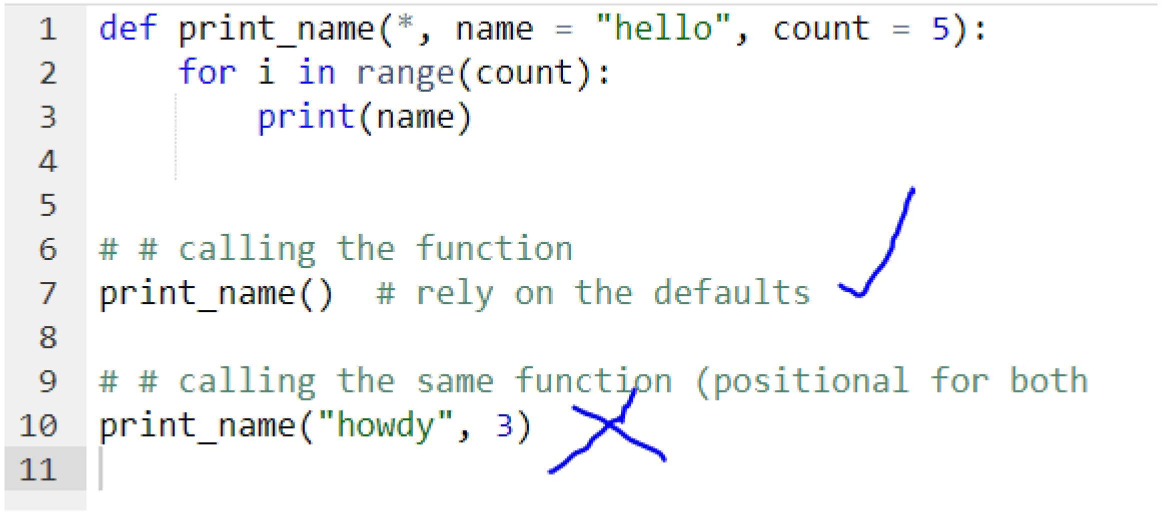
- You can force this by having an extra * argument in the argument list just before the variable names.

- This enforcement will be immensely useful when your function is taking too many arguments with many defaults (for example, see the next slide)

Keyword Arguments (pythontutor)

PYTHON PROGRAMMING by SIVA JASTHI

- You can force this by having an extra * argument in the argument list just before the variable names.
- TypeError: print_name() takes 0 positional arguments but 2 were given
- Python tutor link



```
def print_name(*, name = "hello", count = 5):
    for i in range(count):
        print(name)


# # calling the function
print_name()   # rely on the defaults ✓

# # calling the same function (positional for both
print_name("howdy", 3) ✗

```

- This enforcement will be immensely useful when your function is taking too many arguments with many defaults (for example, see the next slide)

Keyword Arguments (pythontutor)

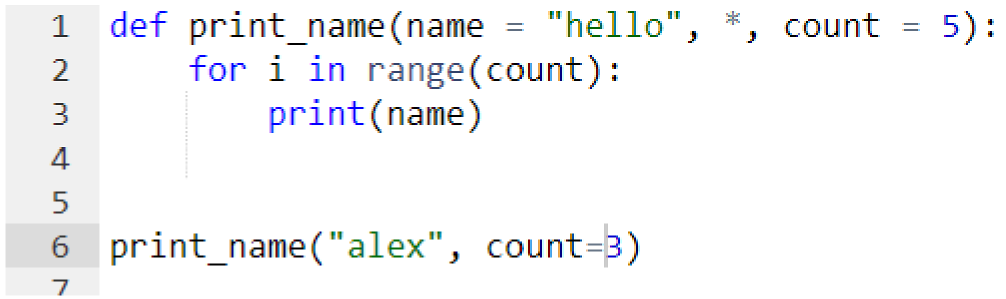
- You can force this by having an extra * argument in the argument list just before the variable names.

- You can keep some arguments as positional and some arguments as mandatory keywords.

- It all depends on where you are placing that extra *

- def fn1(a, b, *, p, q, r)
  - Positional arguments: a,b
  - Keyword arguments: p,q, r

- def fn2(*, a, b, c, d)
  - Positional arguments: NA
  - Keyword arguments: a, b, c, d

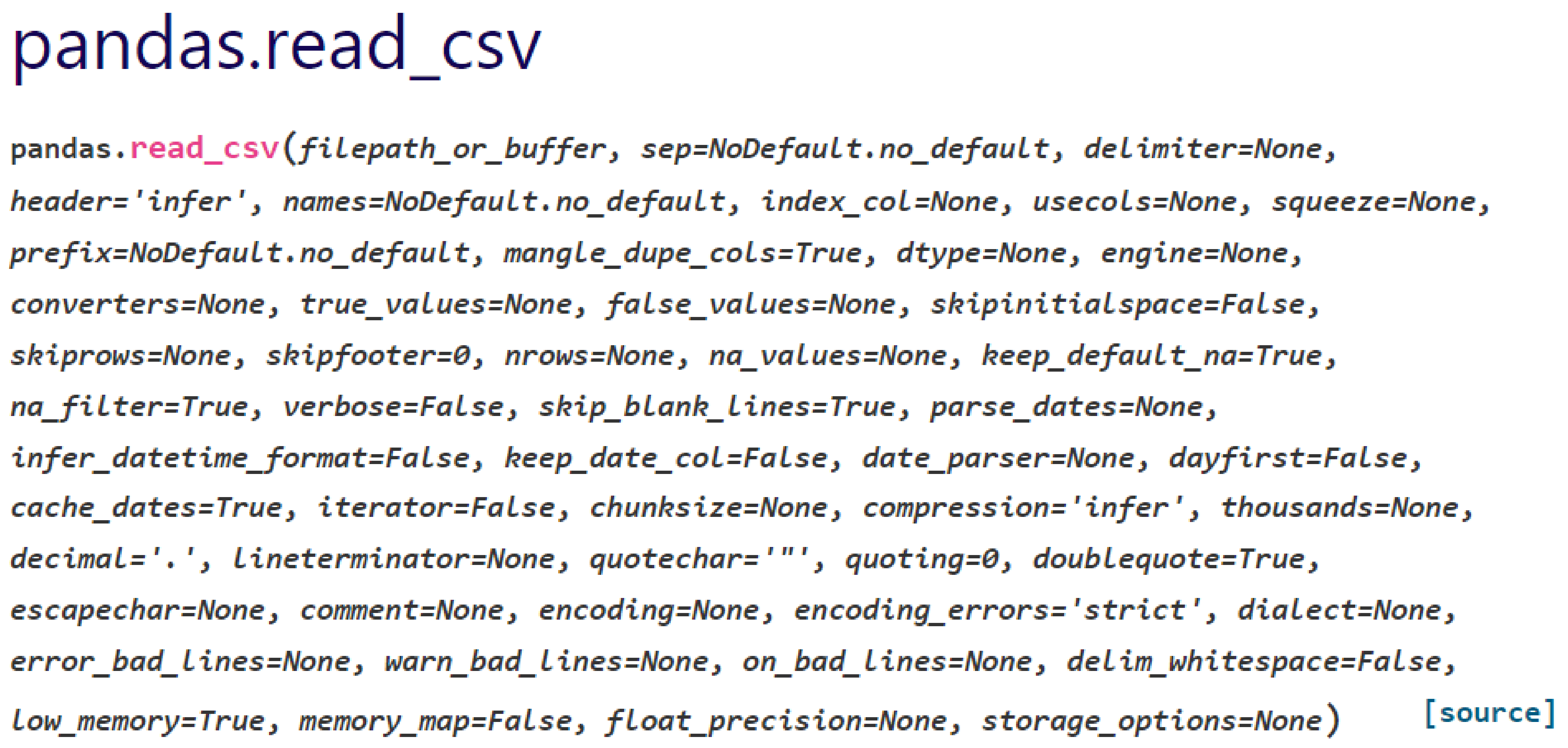


```
def print_name(name = "hello", *, count = 5):
    for i in range(count):
        print(name)


print_name("alex", count=3)
```

Forcing the keyword arguments (link)

PYTHON PROGRAMMING by SIVA JASTHI



pandas.read_csv

```
pandas.read_csv(filepath_or_buffer, sep=NoDefault.no_default, delimiter=None,
header='infer', names=NoDefault.no_default, index_col=None, usecols=None, squeeze=None,
prefix=NoDefault.no_default, mangle_dupe_cols=True, dtype=None, engine=None,
converters=None, true_values=None, false_values=None, skipinitialspace=False,
skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True,
na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=None,
infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None,
decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None,
error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False,
low_memory=True, memory_map=False, float_precision=None, storage_options=None)    [source]
```

Optional Arguments

Mandatory parameters: Explicit values

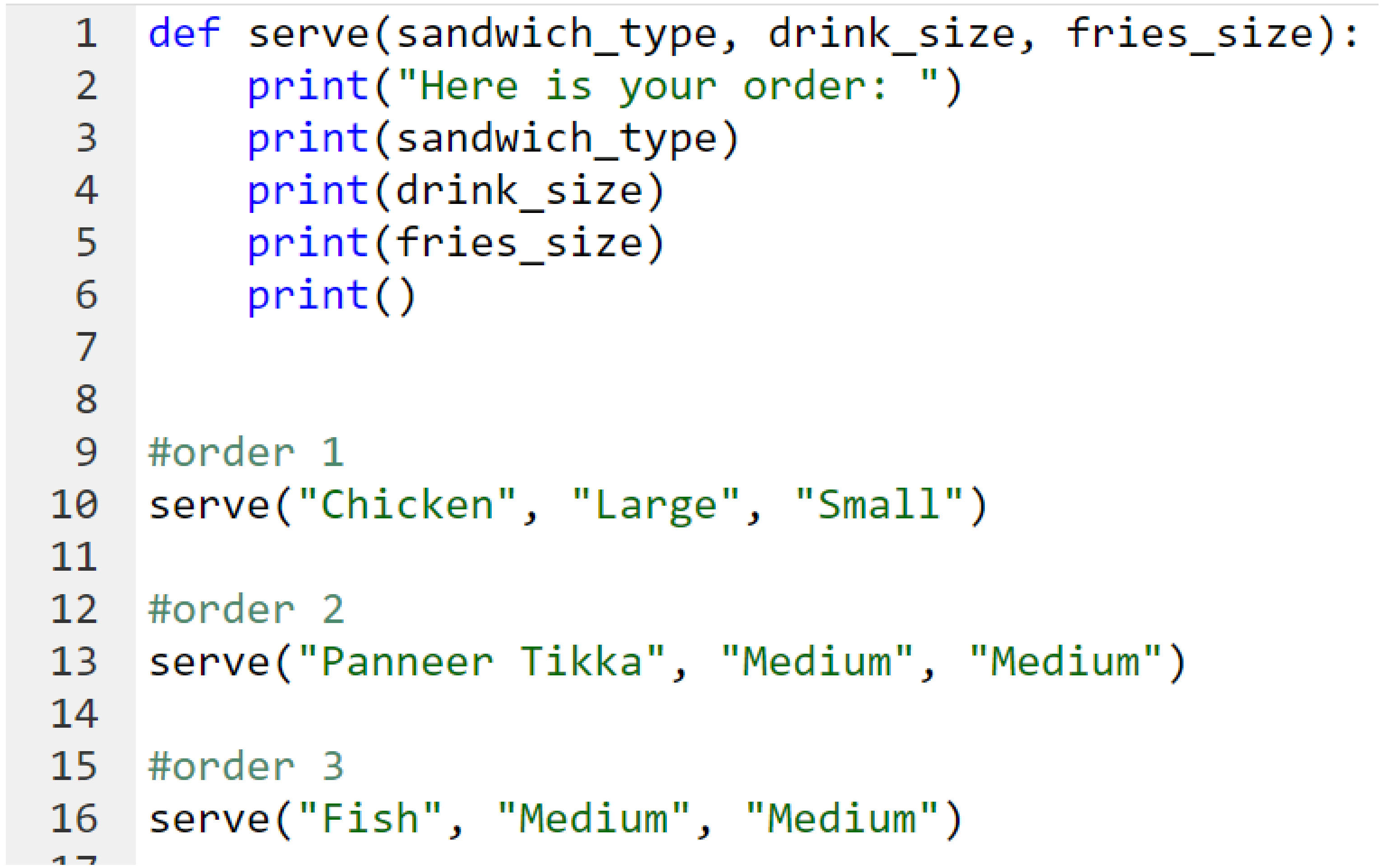
Our function "serve" do not have any "optional" parameters.

All parameters are "mandatory".

Your caller (user, customer) should call your function with explicit order.

The customer can not say "I want a "Fish" sandwich"

Server("Fish") is NOT ok.



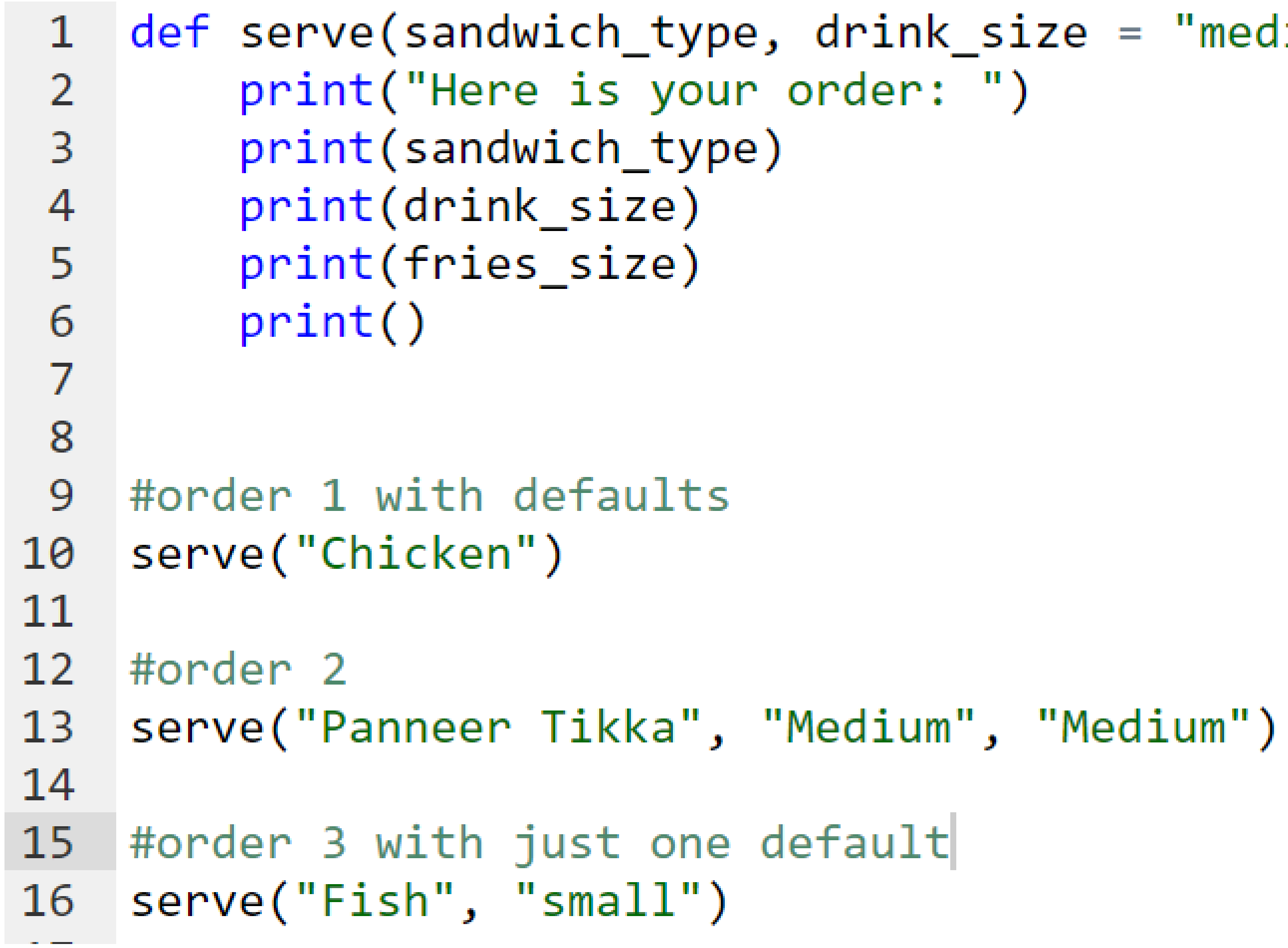
```python
def serve(sandwich_type, drink_size, fries_size):
    print("Here is your order: ")
    print(sandwich_type)
    print(drink_size)
    print(fries_size)
    print()


#order 1
serve("Chicken", "Large", "Small")

#order 2
serve("Panneer Tikka", "Medium", "Medium")

#order 3
serve("Fish", "Medium", "Medium")
```

Optional parameters: Implicit values



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


#order 1 with defaults
serve("Chicken")

#order 2
serve("Panneer Tikka", "Medium", "Medium")

#order 3 with just one default
serve("Fish", "small")
```

Here our function “serve” contains default values for the two parameters.

If your caller provides the values, great! We will use those values.

If your caller does not provide those values, we will use the default values.

This is immensely useful to provide pleasant user experience.

Serve(“Fish”, “medium”, “small”)
Serve(“Fish”, fries_size = “small”)

Optional Parameters

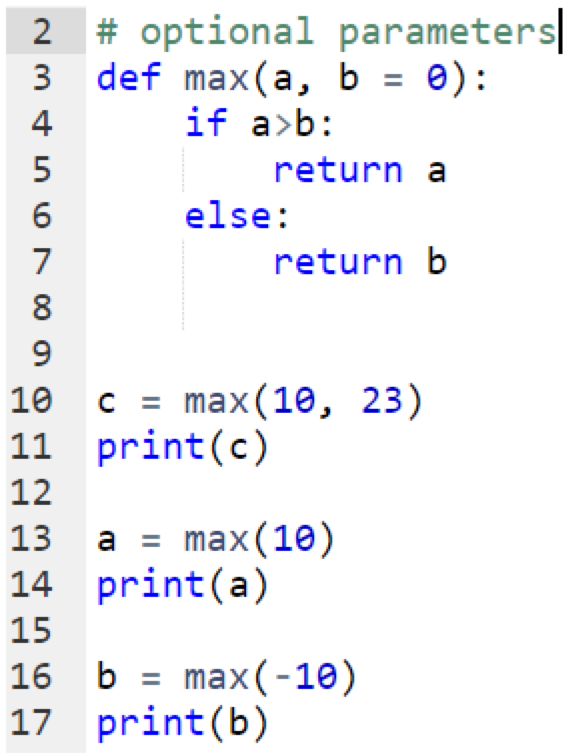
Sometimes it helps to maintain some default values to the formal parameters.

Just imagine that you are a McDonlad employee serving the sandwiches.

Unless the customer orders explicitly, assume that you are going to give “medium” drink and “medium” fries with the sandwich.

Our method takes three parameters:

sandwich_ type
drink_size (default value = “medium”)
fries_size (default value = “medium”)



```
# optional parameters
def max(a, b = 0):
    if a>b:
        return a
    else:
        return b


c = max(10, 23)
print(c)

a = max(10)
print(a)

b = max(-10)
print(b)
```

Variable Number of Arguments

Variable number of arguments (*args)

- (1) function name and (2) number of arguments – are two important things
- What options do we have if we do not know the number of arguments in advance?
- Example:
  - What is the max of 87, 67?
  - What is the max of 87,67, 99?
  - What is the max of 87, 67, 45, 29?
- One option (A) we know is:
  - All the numbers are put into a list.
  - And define a max(list) function that takes just one parameter

Variable number of arguments (*args)

- Another option (B) is:
    - To go with *args
- When python comes across * in front of the argument name, it is treated as a list behind the scenes.

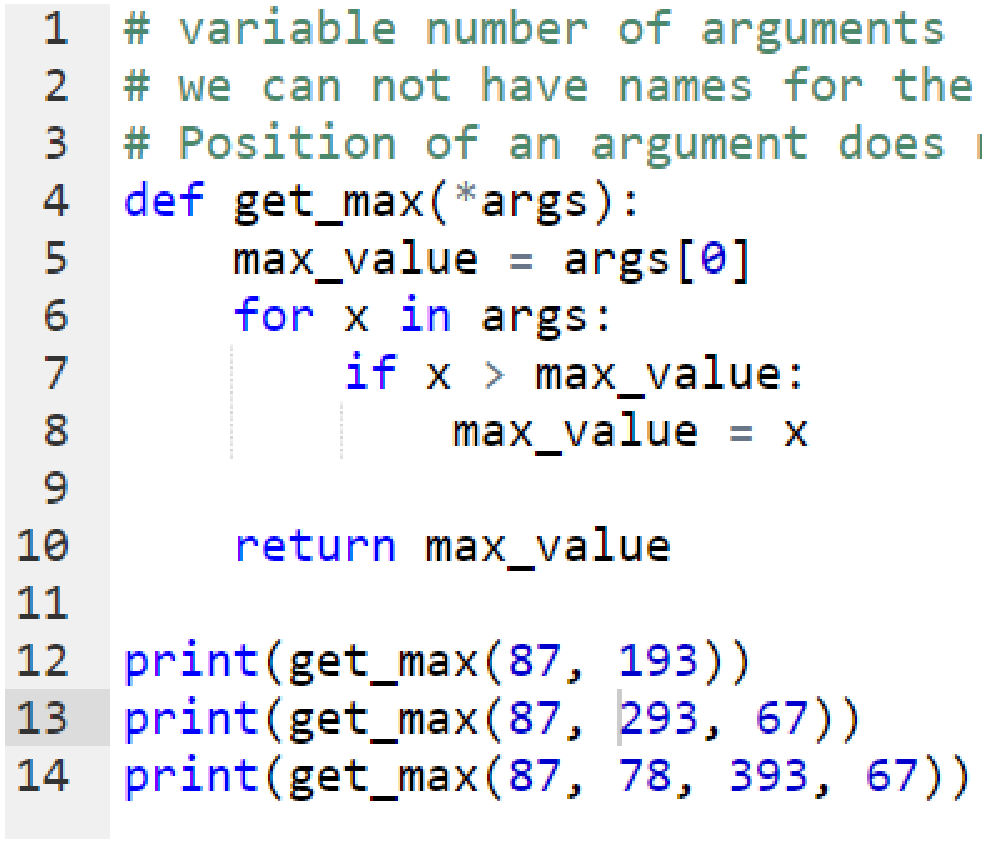
- What is really the difference between Option (A)? And Option (B)?

    - Option (A): you are forcing the user to convert the inputs a list.
    - Option (B): Python is doing it for you

    - Option (A): is how the programmers think
    - **Option (B): is how the end-users think**

Variable number of arguments (*args)

- link



```python
# variable number of arguments
# we can not have names for the varibles
# Position of an argument does not come into play here.
def get_max(*args):
    max_value = args[0]
    for x in args:
        if x > max_value:
            max_value = x

    return max_value

print(get_max(87, 193))
print(get_max(87, 293, 67))
print(get_max(87, 78, 393, 67))
```

Positional Arguments through *args (OK)

- Positional arguments can also be passed through *args
- Because *args is converted a list (formally, we call it an “iterable”), we can assure that there is an “order” to the input
- “order” = “position” is guaranteed.
- def quadratic_equation(*args):

How To Solve It Fast!

$$6x^2 - 17x + 12 = 0$$

$$X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

# Calling the above function (correct way)
quadratic_equation(6, -17, 12)

# Calling the above function (wrong way)
quadratic_equation(12, 6, -17)

Variable number of Keyword Arguments

Variable number of arguments (**kwargs)

- One limitation of *args is: we can not have names (keywords or argument names) for the args list.

- What happens if we want to pass-in ‘name-value’ pairs?

- Example 1:
  - A classroom has N students (we do not know N; it can be 10, 15,20, etc.)
  - Each student has unique ID and marks.
  - Write a function that finds the ID of the student with max marks

- Option A we know is:
  - Send in a dictionary as the parameter

Variable number of arguments (**kwargs)

- Another option (B) is:
  - To go with **kwargs

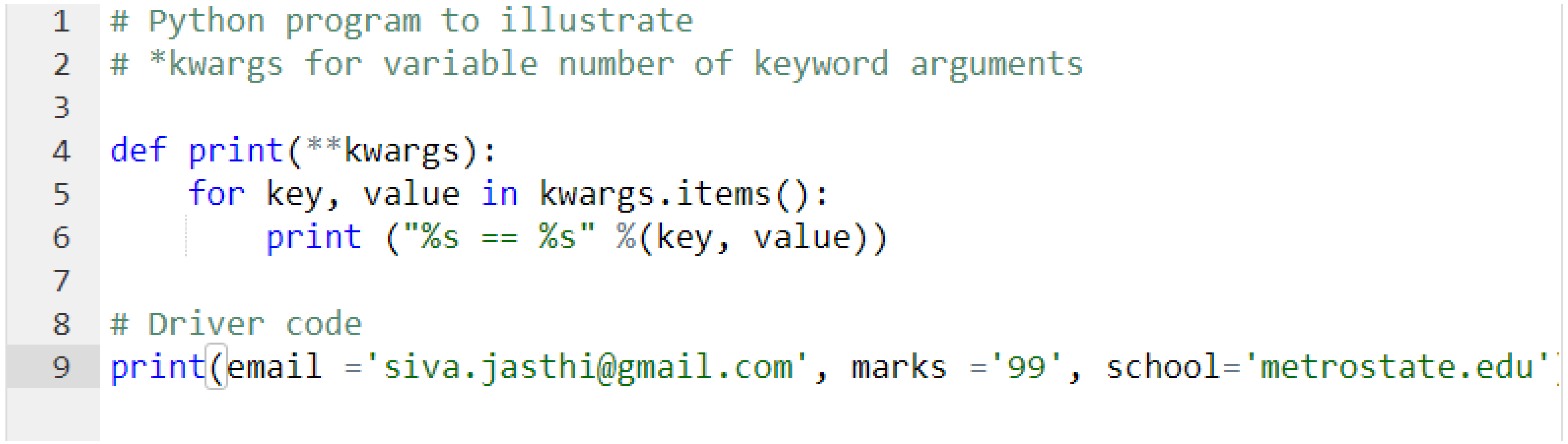
- When python comes across ** in front of the argument name, it is treated as a dictionary behind the scenes.

- Python can traverse the dictionary to get the keywords (names of the parameters) and the corresponding values.

Variable number of Keyword arguments

PYTHON PROGRAMMING by SIVA JASTHI

- link



```
# Python program to illustrate
# *kwargs for variable number of keyword arguments

def print(**kwargs):
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))

# Driver code
print(email ='siva.jasthi@gmail.com', marks ='99', school='metrostate.edu'
```

Functions 2: Summary

- Variable number of arguments *args
- Variable number of key-value pairs **kwargs
- Yield keyword
- Positional arguments (number and position shall match)
- Keyword arguments (variable names = values can appear in any order)
- Keyword arguments can be forced by having an extra * in the list.

- We can also mix and match ‘positional’ arguments and keyword arguments.
  - The placement of * in the argument list dictates the category of the argument

References

Positional and Keyword Arguments:

- https://problemsolvingwithpython.com/07-Functions-and-Modules/07.07-Positional-and-Keyword-Arguments/
- https://www.codeforests.com/2020/11/15/python-positional-keyword-argument/
- https://www.geeksforgeeks.org/args-kwargs-python/

yield keyword (what is difference between yield and return?)
generators (and yield function
https://www.geeksforgeeks.org/python-yield-keyword

Thank You.

PYTHON PROGRAMMING

by SIVA JASTHI