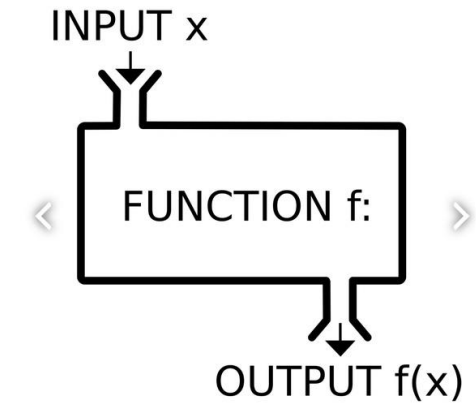| Course	
Term	
Week	
Date	
Chapter. Topic	5. Functions

Positional Arguments

Keyword Arguments

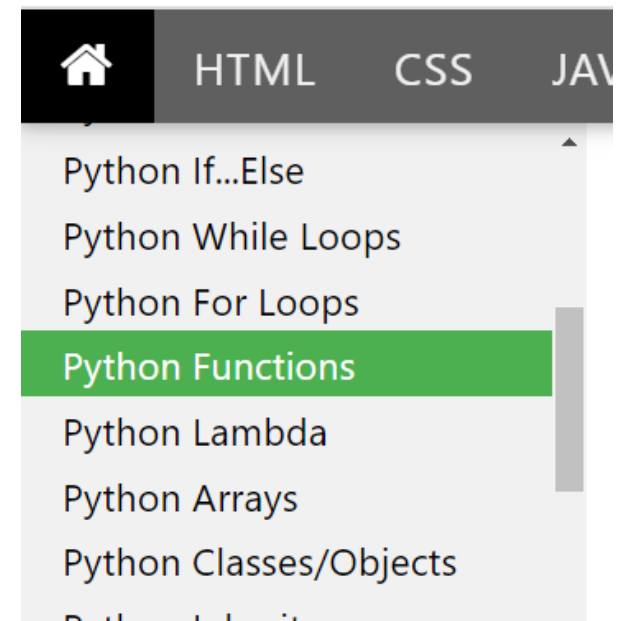**Optional Arguments**

Siva R Jasthi

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.

INPUT x

FUNCTION f:

OUTPUT f(x)

Review and practice "Try It" exercises here

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

HTML   CSS   JAV

Python If...Else
Python While Loops
Python For Loops
Python Functions
Python Lambda
Python Arrays
Python Classes/Objects
Python Inheritance

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)

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

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)

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.

How To Solve It Fast!

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

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

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
1  def print_name(name = "hello", count = 5):
2      for i in range(count):
3          print(name)
4
5
6  # # calling the function
7  # print_name()  # rely on the defaults
8
9  # # calling the same function (positional for both
10 # print_name("howdy", 3)
11
12 # calling the same function (positional for 1; default for 2)
13 #print_name("howdy")
14
15 # calling the same function
16 #print_name(3)  # i want to print hello 3 times
17              # will it print 3 five times?
18              # will it print hello five times
19              # wil it print hello three times
20
21 # how do I call the above function so that I print default hello 3 times?
22 print_name(count=3) # This is the "keyword" argument
23
24 # change the order if I am explicit about the keyword
25 print_name(count=4, name="alex")
```

Keyword Arguments ()

- 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)

- 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

```
1   def print_name(*, name = "hello", count = 5):
2       for i in range(count):
3           print(name)
4
5
6   # # calling the function
7   print_name()   # rely on the defaults     ✓
8
9   # # calling the same function (positional for both
10  print_name("howdy", 3)         ✗
11
```

- 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

```
1  def print_name(name = "hello", *, count = 5):
2      for i in range(count):
3          print(name)
4
5
6  print_name("alex", count=3)
7
```

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

Forcing the keyword arguments (link)

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
1  def serve(sandwich_type, drink_size, fries_size):
2      print("Here is your order: ")
3      print(sandwich_type)
4      print(drink_size)
5      print(fries_size)
6      print()
7
8
9  #order 1
10 serve("Chicken", "Large", "Small")
11
12 #order 2
13 serve("Panneer Tikka", "Medium", "Medium")
14
15 #order 3
16 serve("Fish", "Medium", "Medium")
17
```

Optional parameters: Implicit values

```
 1  def serve(sandwich_type, drink_size = "medium", fries_size = "medium"):
 2      print("Here is your order: ")
 3      print(sandwich_type)
 4      print(drink_size)
 5      print(fries_size)
 6      print()
 7
 8
 9  #order 1 with defaults
10  serve("Chicken")
11
12  #order 2
13  serve("Panneer Tikka", "Medium", "Medium")
14
15  #order 3 with just one default
16  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")

```python
2   # optional parameters
3   def max(a, b = 0):
4       if a>b:
5           return a
6       else:
7           return b
8
9
10  c = max(10, 23)
11  print(c)
12
13  a = max(10)
14  print(a)
15
16  b = max(-10)
17  print(b)
```

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://ww.codeforests.com/2020/11/15/python-positional-keyword-argument/
- https://www.geeksforgeeks.org/args-kwargs-python/

Thank You.