Python Functions

14-2-2024

By Ms. Soofi Shafiya

Function

- *A function is a reusable piece 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.

How to create a function

- "def" keyword is used to define a function in python.

Syntax

```
def <function_name>():
        <function body>
```

Example

```
def my_function():
  x=10
  print("Hello from a function")
  print(x)
```

When you create a variable inside a function, it is local, which means that it only exists inside the function.

How to call a function

Syntax:

<Function_name>()

Example

```
def my_function():
  print("Hello from a function")

my_function()
```

Arguments of a function

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses. We can add as many arguments as we want, just separate them with a comma.

Syntax:

```
def <function_name>(arg1, arg2, …):
        <Function_body>
```

Example

```
def my_function(arg):
  print(arg + "alphabets")

my_function(“abc")
my_function(“xyz")
my_function(“def")
```

Parameters or Arguments?

From a function's perspective:

- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that is sent to the function when it is called.

```
def my_function(arg):
  print(arg + "alphabets")

my_function(“abc")
```

Number of Arguments?

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

*args and **kwargs

Arbitrary Arguments (*args)

- *tuple* of arguments

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Keyword Arguments (kwargs)

We can also send arguments with the key=value syntax. This way the order of the arguments does not matter.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias",
child3 = "Linus")
```

Arbitrary Keyword Arguments (**kwargs)

If number of keyword arguments that will be passed into function is unknown, add ** before the parameter name. This way the function will receive a dictionary of arguments.

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Types of arguments!

Keyword and Positional Argument in Python

```
def myfunc(name, age):
    print("Hi, I am", name)
    print("My age is ", age)

myfunc(name="abc", age=20)

myfunc(age=20, name="abc")
```

```
def myfunc(name, age):
    print("Hi, I am", name)
    print("My age is ", age)

myfunc("abc", 20)

myfunc(20,"abc")
```

```
def minus(a, b):   #function definition
    return a - b


a, b = 20, 10  #variable declaration
result1 = minus(a, b)  #function calling
print("Used Positional arguments:", result1)


#you will get incorrect output because expected was (a-b) but you
will be getting (b-a) because of swapped position of value a and b

result2 = minus(b, a) #function calling
print("Used Positional arguments:", result2)
```

Exercise

Get the correct output using keyword arguments.

Default Parameter Value

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Passing a List as an Argument

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Return Values

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Null Return type

- If we do not specify a return value, the function returns None when it terminates.

```
def f(x):
   x+1 #no return
   if x ==99:
        return

print(f(0))
```

Multiple return values

```
def sum_and_prod(x,y):
   return (x + y,  x * y)
sum_and_prod(5, 6)
```

#tuple is implicitly returned as defined by the use of the commas:
#the parantheses can be omitted

```
def sum_and_prod(x,y):
   return x + y,  x * y
sum_and_prod(5, 6)
```

O/P (11, 30)

Unpacking the returned tuple

s, p = sum_and_prod(5,6)

The pass Statement

```
def myfunction():
  pass
```

Positional-Only Arguments

You can specify that a function can have ONLY positional arguments, or ONLY keyword arguments.

To specify that a function can have only positional arguments, add , / after the argument

```
def my_function(x, /):# you will get an error if you try to send a keyword argument:
  print(x)

my_function(3)
```

Keyword-Only Arguments

To specify that a function can have only keyword arguments, add `*,` before arguments

```
def my_function(*, x):
  print(x)

my_function(x = 3)

def my_function(*, x): #ERROR
  print(x)

my_function(3)
```

Combine Positional-Only and Keyword-Only

```
def my_function(a, b, /, *, c, d):
  print(a + b + c + d)

my_function(5, 6, c = 7, d = 8)
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

Exercise

1. Create a function and print any statement inside function.
2. Create a function with three parameters and return sum of three arguments.
3. Factorial of a number

String Formatting

f-strings

String formatting

Python has ways of creating strings by “filling in the blanks” and formatting them nicely.
This is helpful for when you want to print statements that include variables or statements.
All you need to do is put the letter “f” out the front of your string and then you can include variables with curly-bracket notation {}.

```
Name = “XYZ”
Age = 20
Day = 10
Months=6
Year=2001
Temp1= f”hello, my name is {Name}. I am {Age} years old. I was born {Day}/{Month}/{Year}.”
```