Python Partial Functions

Summary: In this tutorial, you'll learn about Python partial functions and how to define partial functions using the partial function from the functools module.

Introduction to Python partial functions

The following example defines a function (https://www.pythontutorial.net/python-basics/python-functions/) that multiplies two arguments:

```
def multiply(a, b):
    return a*b
```

Sometimes, you just want to multiply an argument with a specified number e.g., 2. To do that, you can reuse the `multiply` function like this:

```
def double(a):
    return multiply(a,2)
```

The `double` function returns the `multiply` function. It passed the number 2 to the second argument of the `multiply` function.

The following shows how to use the `double` function:

```
result = double(10)
print(result)
```

Put it all together:

```
def multiply(a, b):
    return a*b


def double(a):
    return multiply(a, 2)


result = double(10)
print(result)  # 20
```

As you can see, the `double` function reduces the arguments of the `multiply` function.

The `double` function freezes the second argument of the `multiply` function, which results in a new function with a simpler signature.

In other words, `double` function reduces the complexity of the `multiply` function.

In Python, the `double` function is called a partial function.

In practice, you use partial functions when you want to reduce the number of arguments of a function to simplify the function's signature.

Since you'll create partial functions sometimes, Python provides you with the `partial` function from the `functools` standard module to help you define partial functions more easily.

Python partial function from functools module

The following shows the syntax of the `partial` function from the `functools` module:

```
functools.partial(fn, /, *args, **kwargs)
```

The `partial` function returns new `partial` object, which is a callable (https://www.pythontutorial.net/python-built-in-functions/python-callable/) .

When you call the `partial` object, Python calls the `fn` function with the positional arguments `args` and keyword arguments (https://www.pythontutorial.net/python-basics/python-keyword-arguments/) `kwargs` (https://www.pythontutorial.net/python-basics/python-kwargs/) .

The following example shows how to use the `partial` function to define the `double` function from the `multiply` function:

```
from functools import partial

def multiply(a, b):
    return a*b


double = partial(multiply, b=2)

result = double(10)
print(result)
```

Output:

```
20
```

How it works.

- First, import the `partial` function from the `functools` module.
- Second, define the `multiply` function.
- Third, return a partial object from the `partial` function and assign it to the `double` variable.

When you call the `double` , Python calls the `multiply` function where `b` argument defaults to `2` .

If you pass more arguments to a partial object, Python appends them to the `args` argument.

When you pass additional keyword arguments to a partial object, Python extends and overrides the `kwargs` arguments.

Therefore, it's possible to call the `double` like this:

```
double(10, b=3)
```

In this example, Python will call the `multiply` function where the value of the `b` argument is 3, not 2.

And you'll see the following output:

```
30
```

Python partial functions and variables

Sometimes, you may want to use variables for creating partials. For example:

```
from functools import partial


def multiply(a, b):
    return a*b


x = 2
f = partial(multiply, x)

result = f(10)  # 20
print(result)

x = 3
```

```
result = f(10)  # 20
print(result)
```

In this example, we change `x` to `3` and expect that `f(10)` would return `30` instead of 20.

However, `f(10)` returns `20` instead. It's because Python evaluates the value of `x` in the following statement:

```
f = partial(multiply, x)
```

...but not after that, therefore, when `x` references (https://www.pythontutorial.net/advanced-python/python-references/) to the new number (`3`), the partial function doesn't change.

Summary

- Use `partial` function from the `functools` module to create partial functions in Python.