Introduction to Programming and Data Structures

Python – Functions

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

August, 2023

Functions

```
def <function-name>(<argument 1>, ..., <argument n>):
    Statement 1
    Statement 2
    Statement 3
    return <expression>
```

Functions

```
def <function-name>(<argument 1>, ..., <argument n>):
    Statement 1
    Statement 2
    Statement 3
    return <expression>
```

Note: You may either return a values obtained from an `<expression>` or return nothing based on your requirement.

Functions – Understanding the internals

```
def function(a, b, c):
    print('Inside 1:', hex(id(a)), hex(id(b)), hex(id(c)))
    b += 10
    print('Inside 2:', hex(id(a)), hex(id(b)), hex(id(c)))
x, y, z = 10, 10, 20
print('Outside:', hex(id(x)), hex(id(y)), hex(id(z)))
function(x, y, z)
```

Functions – Understanding the internals

```
def function(a, b, c):
    print('Inside 1:', hex(id(a)), hex(id(b)), hex(id(c)))
    b += 10
    print('Inside 2:', hex(id(a)), hex(id(b)), hex(id(c)))
x, y, z = 10, 10, 20
print('Outside:', hex(id(x)), hex(id(y)), hex(id(z)))
function(x, y, z)
```

Output:

```
Outside: 0x7fc4436a0210 0x7fc4436a0210 0x7fc4436a0350
Inside 1: 0x7fc4436a0210 0x7fc4436a0210 0x7fc4436a0350
Inside 2: 0x7fc4436a0210 0x7fc4436a0350 0x7fc4436a0350
```

Comparing the lengths of two positive integers

```
def compare(m, n):
    while m and n:
        m //= 10
        n //= 10
    return m-n
x, y = input('Enter two numbers: ').split()
f = compare(int(x), int(y))
if f > 0:
    print(x, 'has larger number of digits')
elif f < 0:
    print(y, 'has larger number of digits')
else:
    print('Both have same number of digits')
```

Finding prefixes of a string

```
def prefix(str):
    start, end = 0, 0
    while start < len(str):
        if str[end] == str[end-start]:
            print(str[start:end + 1], end= " ")
            end += 1
            if end == len(str):
                start += 1
                end = start
        else:
            start += 1
            end = start # Index of substring
```

Input: prefix('Python')

Lambda functions

Python provides an anonymous function `lambda` that can take any number of arguments, but can only have one expression.

```
x = lambda a: a + 10
print(x(5))
```

Here, the output will be 15.

Note: Lambda functions execute comparatively faster (in general) because they are inline functions.

Lambda functions

```
a, b = 3, 5
maximum = lambda a, b: a if a > b else b
print(f'{maximum(a,b)} is the maximum among', a, 'and', b)
```

Lambda functions

```
a, b = 3, 5
maximum = lambda a, b: a if a > b else b
print(f'{maximum(a,b)} is the maximum among', a, 'and', b)
```

Output:

```
5 is the maximum among 3 and 5
```

Lambda functions

```
l = lambda n: n
q = lambda n: n ** 2
c = lambda n: n ** 3
x = 2
y = c(x) + 3*q(x) + 2*l(x) + 1 # y = x^3 + 3x^2 + 2x + 1
print(y)
```

Lambda functions

```
l = lambda n: n
q = lambda n: n ** 2
c = lambda n: n ** 3
x = 2
y = c(x) + 3*q(x) + 2*l(x) + 1 # y = x^3 + 3x^2 + 2x + 1
print(y)
```

Output:

```
25
```

Late binding closures

Python's closures are late binding. Hence, the values of variables used in closures are looked up when the inner function is called.

```
def increase():
    return [lambda x : i + x for i in range(5)]
for add in increase():
    print(add(10))
```

Here, whenever any of the returned functions are called, the value of i is looked up in the surrounding scope at call time. By then, the loop has completed and i is left with its final value of 4. So, it will print {14, 14, 14, 14, 14} instead of {10, 11, 12, 13, 14}.

Problems

1. A number is PSEUDOPERFECT if the sum of all or some of its proper divisors is equal to the number itself. Write a program to verify whether a given number is pseudoperfect or not. E.g., 12 is a pseudoperfect number from its divisors 1, 2, 3, 4 and 6, we can ignore 4 and derive $1 + 2 + 3 + 6 = 12$.
2. Write a program to multiply a pair of matrices given as user input employing the standard approach. Keep a check whether the input matrices are multipliable or not. Further try implementing the Strassen's matrix multiplication algorithm.

Problems

3. Write a program to apply the Pollard's rho algorithm (Pollard, BIT Numerical Mathematics, 1975) on a semi-prime number, say `n = pq`, to find its non-trivial prime factors `p` and `q`. The Pollard's rho algorithm works as follows:

```
x ← 2, y ← 2, d ← 1
while d = 1:
    x ← g(x)
    y ← g(g(y))
    d ← gcd(|x - y|, n)
if d = n:
    return failure
else:
    return d
```

Link to the paper: `https://link.springer.com/article/10.1007/BF01933667`