

Functions

Motivation: We often want to re-use code blocks

```
In [1]: sentence = "Time flies like an arrow."
        # first we count, using a dict
        counts = {} # empty dict
        for c in sentence:
            if c in counts: # have we seen this letter before?
                counts[c]+=1 # yes, increase count by 1
            else:
                counts[c]=1      # no, set count to 1

        # now that we have the counts, we print them
        for c in counts:      # loop over all letters in the dict
            print(c, "appears", counts[c], "times.")
```

Motivation: We often want to re-use code blocks

```
In [1]: sentence = "Time flies like an arrow."
```

```
# first we count, using a dict
```

```
counts = {} # empty dict
```

```
for c in sentence:
```

```
    if c in counts: # have we seen this letter before?
```

```
        counts[c]+=1 # yes, increase count by 1
```

```
    else:
```

```
        counts[c]=1 # no, set count to 1
```

```
# now that we have the counts, we print them
```

```
for c in counts: # loop over all letters in the dict
```

```
    print(c, "appears", counts[c], "times.")
```

generic
letter-
counting
code

Motivation: We often want to re-use code blocks

```
In [1]: sentence = "Time flies like an arrow."
```

```
# count letters using function
```

```
counts = count_letters(sentence)  
          function
```

```
# now that we have the counts, we print them
```

```
for c in counts:      # loop over all letters in the dict  
    print(c, "appears", counts[c], "times.")
```

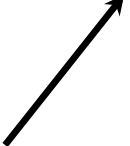
Code for letter-counting function

```
In [1]: def count_letters(str):  
        counts = {} # empty dict  
        for c in str:  
            if c in counts: # does letter exist in dict?  
                counts[c]+=1 # yes, increase count by 1  
            else:  
                counts[c]=1 # no, set count to 1  
        return counts # return result
```

Code for letter-counting function

```
In [1]: def count_letters(str):  
        counts = {} # empty dict  
        for c in str:  
            if c in counts: # does letter exist in dict?  
                counts[c] += 1 # yes, increase count by 1  
            else:  
                counts[c] = 1 # no, set count to 1  
        return counts # return result
```

keyword indicating
function definition



Code for letter-counting function

```
In [1]: def count_letters(str):  
        counts = {} # empty dict  
        for c in str:  
            if c in counts: # does letter exist in dict?  
                counts[c] += 1 # yes, increase count by 1  
            else:  
                counts[c] = 1 # no, set count to 1  
        return counts # return result
```

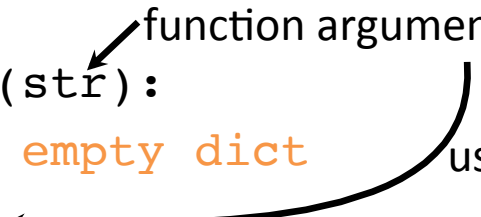
function argument

Code for letter-counting function

```
In [1]: def count_letters(str):  
        counts = {} # empty dict  
        for c in str:  
            if c in counts: # does letter exist in dict?  
                counts[c] += 1 # yes, increase count by 1  
            else:  
                counts[c] = 1 # no, set count to 1  
        return counts # return result
```

function argument

used here

A diagram consisting of two arrows. The first arrow points from the text 'function argument' to the parameter 'str' in the function definition 'def count_letters(str):'. The second arrow starts from the parameter 'str' and curves around to point at the 'str' in the 'for c in str:' loop, indicating its use as a function argument.

Code for letter-counting function

```
In [1]: def count_letters(str):  
        counts = {} # empty dict  
        for c in str:  
            if c in counts: # does letter exist in dict?  
                counts[c] += 1 # yes, increase count by 1  
            else:  
                counts[c] = 1 # no, set count to 1  
        return counts # return result
```

↑
function returns its result here

General form for function definitions

```
def name(argument1, argument2, ...):  
    code, making use of variables  
    argument1, argument2, etc  
return result
```

Indentation determines which lines belong to a function

```
def f():  
    print("A")    # part of function  
    print("B")    # part of function  
    print("C")    # part of function  
print("D")        # not part of function
```

Note: A **return** statement is not required in a function definition.

Important guidelines for writing functions

- You can never write too many functions
- If your code doesn't fit on your screen, or uses more than 3 levels of indentation, break it into functions