

Python

Functions

A programming language should *not* include everything anyone might ever want

A programming language should *not* include
everything anyone might ever want
Instead, it should make it easy for people to create
what they need to solve specific problems

A programming language should *not* include
everything anyone might ever want

Instead, it should make it easy for people to create
what they need to solve specific problems

Define functions to create higher-level operations

A programming language should *not* include everything anyone might ever want

Instead, it should make it easy for people to create what they need to solve specific problems

Define functions to create higher-level operations

"Create a language in which the solution to your original problem is trivial."

Define functions using `def`

Define functions using def

```
def greet():  
    return 'Good evening, master'
```



Define functions using def

```
def greet():  
    return 'Good evening, master'  
  
temp = greet()  
print(temp)  
Good evening, master
```



Give them parameters

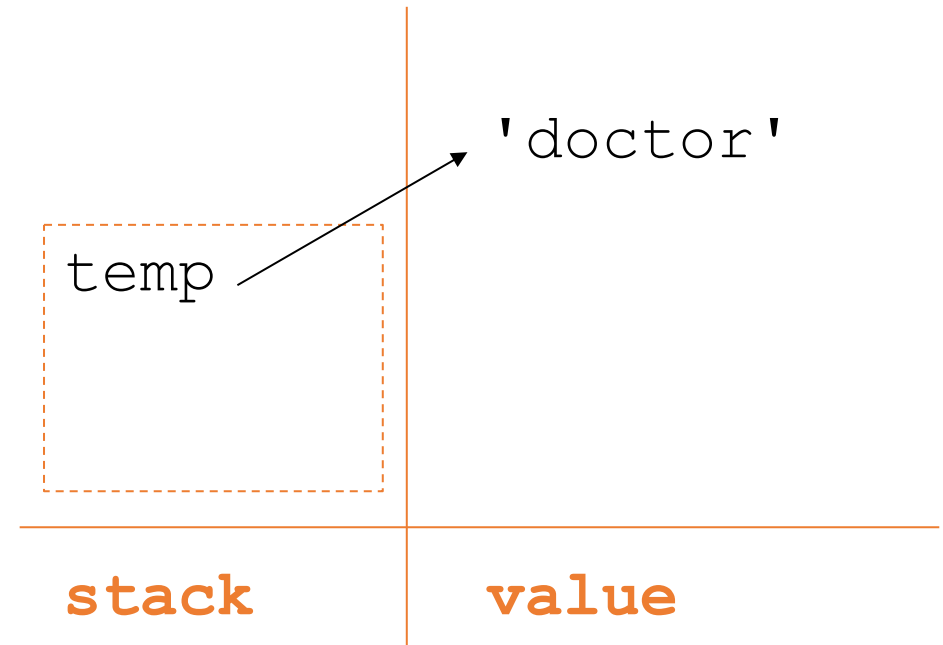
Give them parameters

```
def greet(name):  
    answer = 'Hello, ' + name  
    return answer
```

Give them parameters

```
def greet(name):  
    answer = 'Hello, ' + name  
    return answer
```

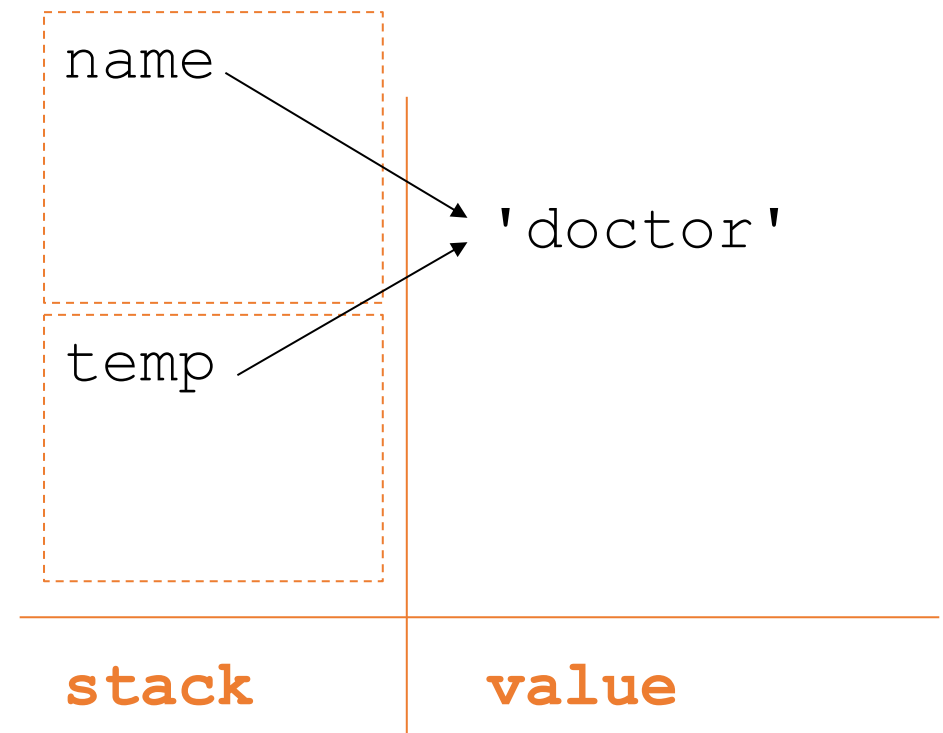
```
temp = 'doctor'
```



Give them parameters

```
def greet(name):  
    answer = 'Hello, ' + name  
    return answer
```

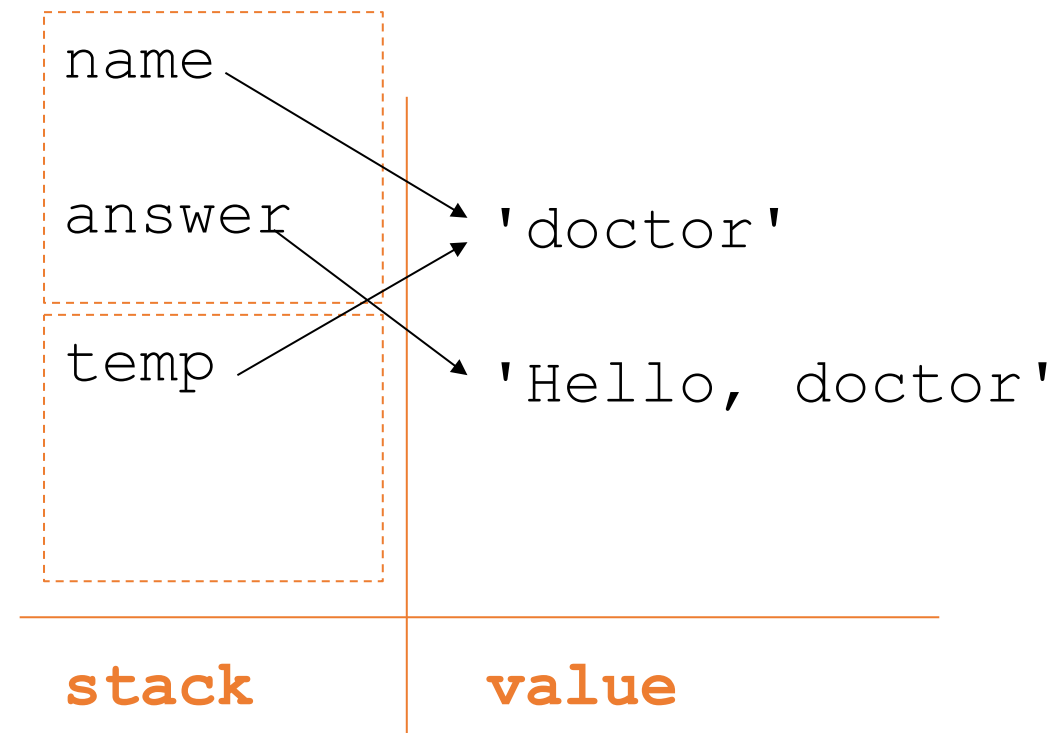
```
temp = 'doctor'  
result = greet(temp)
```



Give them parameters

```
def greet(name):  
    answer = 'Hello, ' + name  
    return answer
```

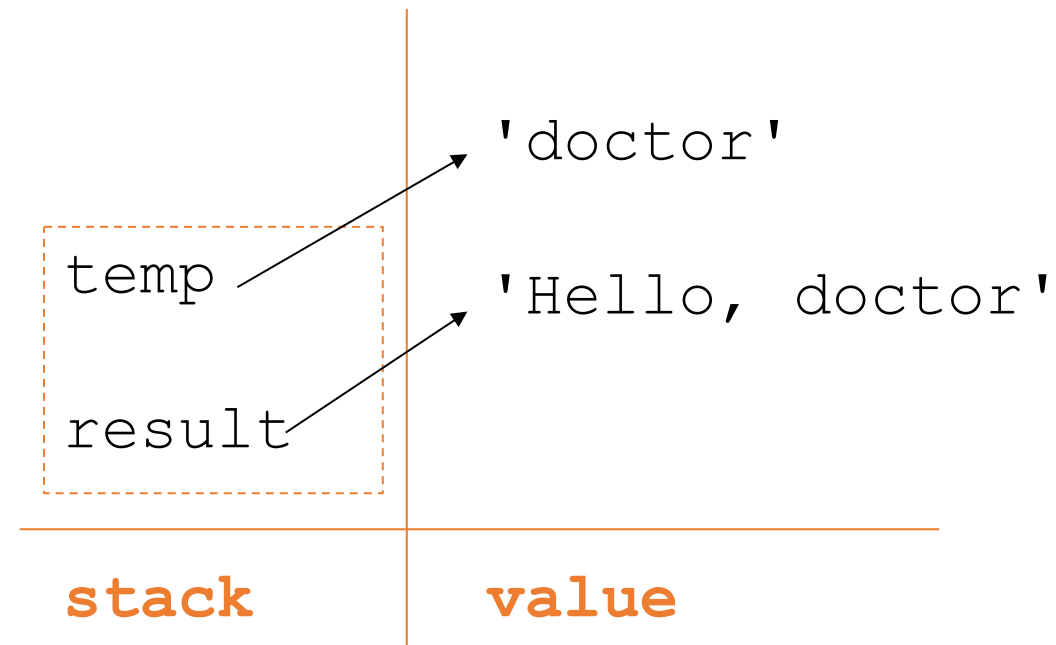
```
temp = 'doctor'  
result = greet(temp)
```



Give them parameters

```
def greet(name):  
    answer = 'Hello, ' + name  
    return answer
```

```
temp = 'doctor'  
result = greet(temp)
```



Only see variables in the *current* and *global* frames

Only see variables in the *current* and *global* frames

Current beats global

Only see variables in the *current* and *global* frames

Current beats global

```
def greet(name):  
    temp = 'Hello, ' + name  
    return temp
```

```
temp = 'doctor'  
result = greet(temp)
```

Can pass values in and accept results directly

Can pass values in and accept results directly

```
def greet(name):  
    return 'Hello, ' + name  
  
print(greet('doctor'))
```

Can return at any time

Can return at any time

```
def sign(num) :  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    else:  
        return -1
```

Can return at any time

```
def sign(num):
```

```
    if num > 0:
```

```
        return 1
```



```
    elif num == 0:
```

```
        return 0
```

```
    else:
```

```
        return -1
```

```
print(sign(3))
```

1

Can return at any time

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    else:  
        return -1
```



```
print(sign(3))
```

1

```
print(sign(-9))
```

-1

Can return at any time

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    else:  
        return -1
```

```
print(sign(3))
```

1

```
print(sign(-9))
```

-1

Over-use makes functions
hard to understand

Can return at any time

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    else:  
        return -1
```

```
print(sign(3))
```

1

```
print(sign(-9))
```

-1

Over-use makes functions

hard to understand

No prescription possible, but:

Can return at any time

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    else:  
        return -1
```

```
print(sign(3))
```

1

```
print(sign(-9))
```

-1

Over-use makes functions
hard to understand

No prescription possible, but:

- a few at the beginning
to handle special cases

Can return at any time

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    else:  
        return -1
```

```
print(sign(3))
```

1

```
print(sign(-9))
```

-1

Over-use makes functions
hard to understand

No prescription possible, but:

- a few at the beginning
to handle special cases
- one at the end for the
"general" result

Every function returns something

Every function returns something

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    # else:  
    #     return -1
```

Every function returns something

```
def sign(num) :  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    # else:  
    #     return -1  
  
print(sign(3))  
1
```

Every function returns something

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    # else:  
    #     return -1
```

```
print(sign(3))
```

1

```
print(sign(-9))
```

None

Every function returns something

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    # else:  
    #     return -1
```

```
print(sign(3))
```

1

```
print(sign(-9))
```

None

If the function doesn't return
a value, Python returns None

Every function returns something

```
def sign(num):  
    if num > 0:  
        return 1  
    elif num == 0:  
        return 0  
    # else:  
    #     return -1  
  
print(sign(3))  
1  
print(sign(-9))  
None
```

If the function doesn't return
a value, Python returns `None`
Yet another reason why
commenting out blocks of code
is a bad idea...

Functions and parameters don't have types

Functions and parameters don't have types

```
def double(x):  
    return 2 * x
```

Functions and parameters don't have types

```
def double(x):  
    return 2 * x
```

```
print(double(2))  
4
```

Functions and parameters don't have types

```
def double(x):  
    return 2 * x
```

```
print(double(2))
```

4

```
print(double('two'))
```

twotwo

Functions and parameters don't have types

```
def double(x):  
    return 2 * x  
  
print(double(2))  
4  
print(double('two'))  
twotwo
```

Only use this when the
function's behavior depends
only on properties that all
possible arguments share

Functions and parameters don't have types

```
def double(x):  
    return 2 * x
```

```
print(double(2))  
4
```

```
print(double('two'))  
twotwo
```

Only use this when the function's behavior depends *only* on properties that all possible arguments share

```
if type(arg) == int:  
    ...  
elif type(arg) == str:  
    ...  
...
```

Can define *default parameter values*

Can define *default parameter values*

```
def adjust(value, amount=2.0) :  
    return value * amount
```

Can define *default parameter values*

```
def adjust(value, amount=2.0):  
    return value * amount
```

```
print(adjust(5))  
10.0
```

Can define *default parameter values*

```
def adjust(value, amount=2.0):  
    return value * amount
```

```
print(adjust(5))
```

```
10.0
```

```
print(adjust(5, 1.001))
```

```
5.004999999999999
```

"When should I write a function?"

"When should I write a function?"

Human short term memory can hold 7 ± 2 items

"When should I write a function?"

Human short term memory can hold 7 ± 2 items

If someone has to keep more than a dozen things
in their mind at once to understand a block of code,
it's too long

"When should I write a function?"

Human short term memory can hold 7 ± 2 items

If someone has to keep more than a dozen things
in their mind at once to understand a block of code,
it's too long

Break it into comprehensible pieces with functions

"When should I write a function?"

Human short term memory can hold 7 ± 2 items

If someone has to keep more than a dozen things
in their mind at once to understand a block of code,
it's too long

Break it into comprehensible pieces with functions

Even if each function is only called once

"When should I write a function?"

Human short term memory can hold 7 ± 2 items

If someone has to keep more than a dozen things
in their mind at once to understand a block of code,
it's too long

Break it into comprehensible pieces with functions

Even if each function is only called once

If a piece of code would be copied multiple times:
definitely use a function!

Python

More on Functions

You can assign a function to a variable

```
def threshold(signal):  
    return 1.0 / sum(signal)
```

```
t = threshold  
print(t([0.1, 0.4, 0.2]))  
1.4285714285714286
```

Can put (a reference to) the function in a list

```
def area(r):  
    return PI * r * r  
  
def circumference(r):  
    return 2 * PI * r  
  
funcs = [area, circumference]  
  
for f in funcs:  
    print(f(1.0))  
3.14159  
6.28318
```

Can pass (a reference to) the function into a function

```
def call_it(func, value):  
    return func(value)
```

```
print(call_it(area, 1.0))  
3.14159
```

```
print(call_it(circumference, 1.0))  
6.28318
```

Must need to know *something* about the function
in order to call it

Must need to know *something* about the function
in order to call it
Like number of arguments

Must need to know *something* about the function
in order to call it

~~Like number of arguments~~

Must need to know *something* about the function
in order to call it

~~Like number of arguments~~

```
def add_all(*args):  
    total = 0  
    for a in args:  
        total += a  
    return total
```

Must need to know *something* about the function
in order to call it

~~Like number of arguments~~

```
def add_all(*args):  
    total = 0  
    for a in args:  
        total += a  
    return total
```

Must need to know *something* about the function
in order to call it

~~Like number of arguments~~

```
def add_all(*args):  
    total = 0  
    for a in args:  
        total += a  
    return total
```

```
print(add_all())
```

0

Must need to know *something* about the function
in order to call it

~~Like number of arguments~~

```
def add_all(*args):  
    total = 0  
    for a in args:  
        total += a  
    return total  
  
print(add_all())  
0  
print(add_all(1, 2, 3))  
6
```

Connecting functions and sequences (1)

List Comprehensions come in handy:

1. Send a sequence to a function to create a new sequence of only positive numbers:

```
def positive(x):  
    return x >= 0
```

```
print([x for x in [-3, -2, 0, 1, 2] if positive(x)])  
[0, 1, 2]
```

Connecting functions and sequences (2)

2. Send a sequence of numbers to a function that will return the negative value of each item:

```
def negate(x):  
    return -x
```

```
print([negate(x) for x in [-3, -2, 0, 1, 2]])  
[3, 2, 0, -1, -2]
```