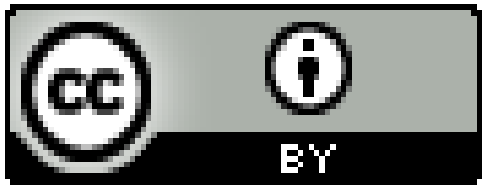
Python

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

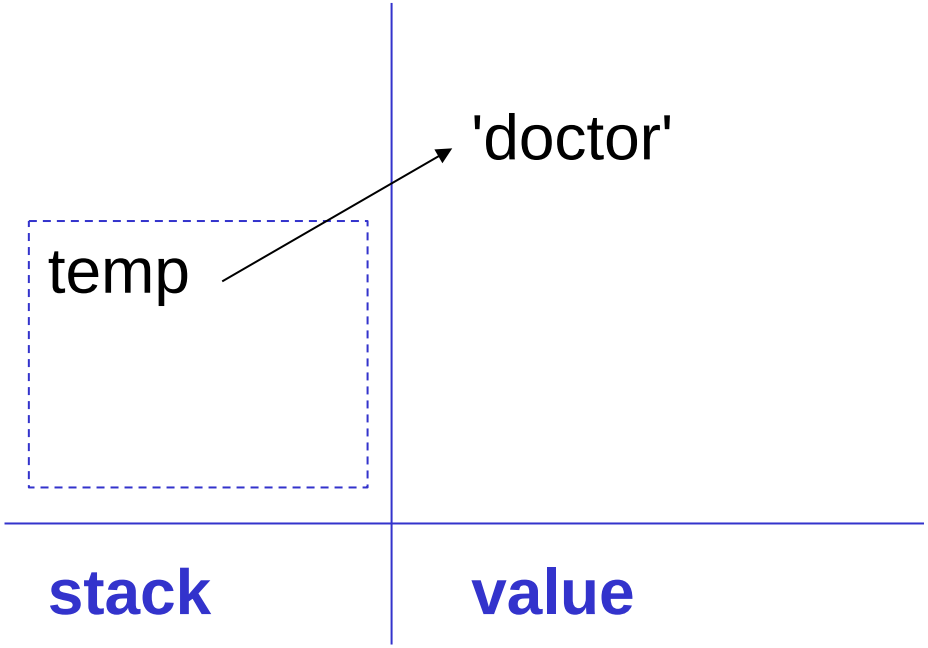
Give them parameters

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

Give them parameters

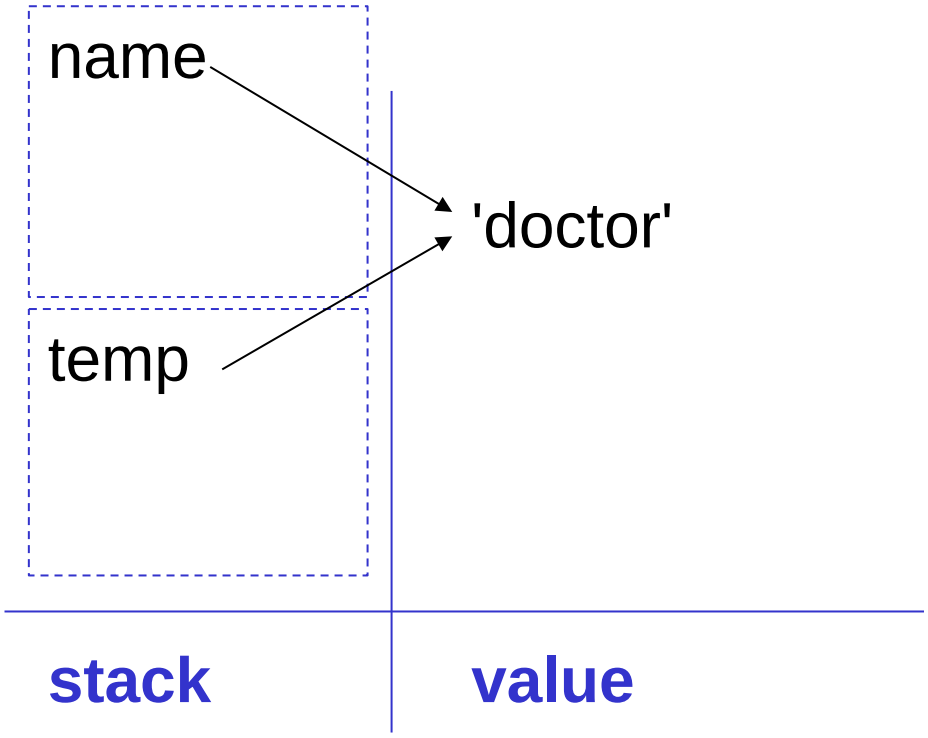
```
def greet(name):
  answer = 'Hello, ' + name
  return answer
```

```
temp = 'doctor'
```



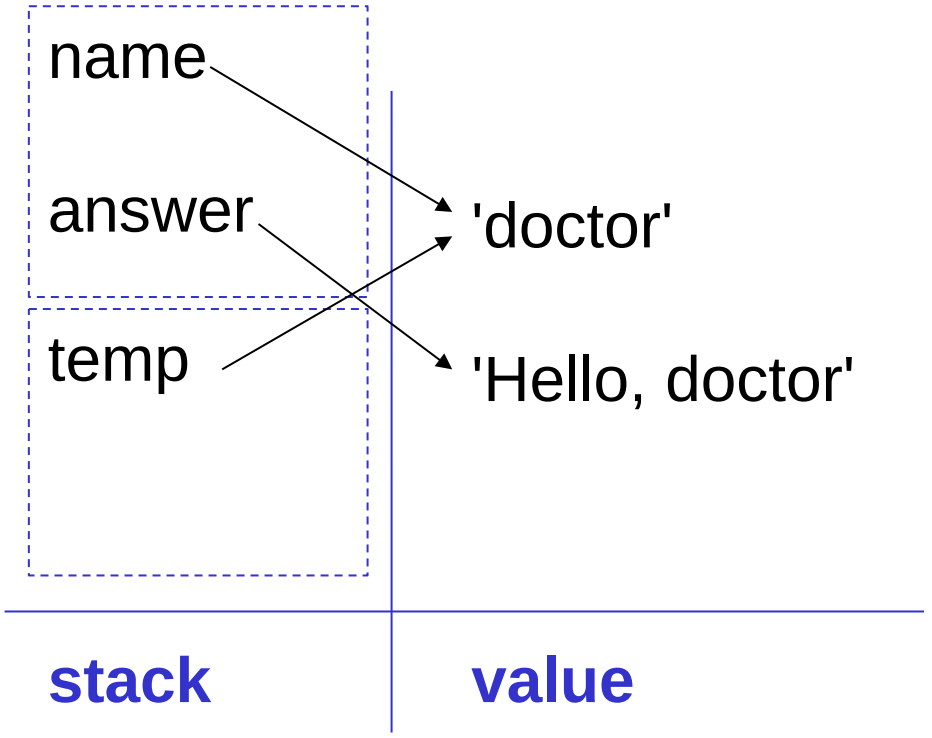
Give them parameters

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

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



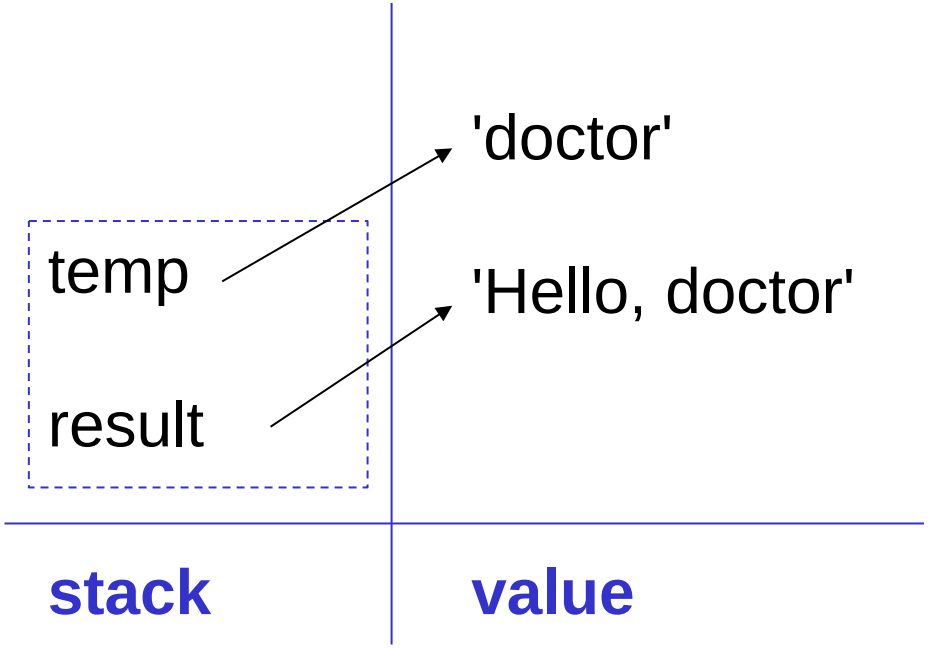
Give them parameters

```
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

Can define *default parameter values*

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

print adjust(5)
10
print adjust(5, 1.001)
5.005
```

"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