Functions and Parameters
CS106A, Stanford University
Learning Goals

1. Get more practice with function parameters
2. Understand information flow in a program
3. Learn about Python's doctest feature
def main():
    avg = average(5.0, 10.2)
    print(avg)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    avg = average(5.0, 10.2)
    print(avg)

def average(a, b):
    sum = a + b
    return sum / 2
Parameters

Parameters let you provide a function with some information when you are calling it.
# Constant - visible to all functions
MAX_NUM = 4

def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
# Constant - visible to all functions
MAX_NUM = 4

def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
Understand the mechanism
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

i 0
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

    i = 0
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

n 0  result 1  i
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i

    return result

n 0  result 1  i 1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0 1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

i 1

0 1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0 1
```python
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

```
0  1
```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

0 1
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

n  1 result 1 i
```python
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

```
1 1 1
1 1
0 1
```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

0 1
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

0 1
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0 1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

print(0, factorial(0))
print(1, factorial(1))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0 1
1 1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0  1
1  1
```python
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

```
0 1
1 1
```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0 1
1 1
2 2
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0  1
1  1
2  2
3  3
def main():
    for i in range(MAX NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0 1
1 1
2 2
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0  1
1  1
2  2
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0 1
1 1
2 2

6
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0  1
1  1
2  2
3  6
def main():
    for i in range(MAX NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0 1
1 1
2 2
3 6
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

Done!

0 1
1 1
2 2
3 6
Parameters

Every time a function is called, new memory is created for that call. Parameter values are passed in.

All local variables start fresh (no old values)
An interlude: doctest
def factorial(n):
    """
    This function returns the factorial of n
    Input: n (number to compute the factorial of)
    Returns: value of n factorial
    Doctests:
    >>> factorial(3)
    6
    >>> factorial(1)
    1
    >>> factorial(0)
    1
    """
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def factorial(n):
    """
    This function returns the factorial of n
    Input: n (number to compute the factorial of)
    Returns: value of n factorial
    Doctests:
    >>> factorial(3)
    6
    >>> factorial(1)
    1
    >>> factorial(0)
    1
    """

    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

Say this was in file "fact.py"
To run doctests (on PC):
> py -m doctest fact.py -v
Testing: **Why** doesn’t your program work?
Let's try it!!
# NOTE: This program is buggy!!

def add_five(x):
    x += 5

def main():
    x = 3
    add_five(x)
    print("x = " + str(x))
# NOTE: This program is **buggy**!!

```python
def add_five(x):
    x += 5

def main():
    x = 3
    add_five(x)
    print("x = " + str(x))
```

---

**Bad Times With functions**
# NOTE: This program is feeling just fine...

def add_five(x):
    x += 5
    return x

def main():
    x = 3
    x = add_five(x)
    print("x = " + str(x))

Good Times With functions
# NOTE: This program is feeling just fine...

def add_five(x):
    x += 5
    return x

def main():
    x = 3
    x = add_five(x)
    print("x = " + str(x))

When we want to “reassign” x inside a helper function, employ the x = change(x) pattern!
Parameter passing mechanism

When a parameter is passed during a function call, a new variable is created for the lifetime of the function call.

That new variable may or may not have the same name as the value that was passed in!
# NOTE: This program is ** buggy!!**

```python
def add_five(x):
    x += 5

def main():
    x = 3
    add_five(x)
    print("x = " + str(x))
```

These are two separate variables. They are not linked!

Only relationship: value of main’s x is used when creating add_five’s x
Later on in class... we will see cases where changes to variables in helper functions seem to persist! It will be great. We will let you know when we get there and exactly when that happens!
Careful!
def main():
    print("hello world")

def say_goodbye():
    print("goodbye!")

Technically legal, but often a sign at the start that you are confusing function definition and function call.
def main():
    print("hello world")
    say_goodbye()

def say_goodbye():
    print("goodbye!")
Learning Goals

1. Get more practice with function parameters
2. Understand information flow in a program
3. Learn about Python's doctest feature
The Whole Burrito:
calendar.py