

# Functions

- Functions are groups of statements to which you give a name.
  - ***Defining*** a function uses the "def" keyword.
- That group of statements can then be referred to by that name later in the program.
  - ***Calling*** a function uses the name of the function then an opening/closing set of parentheses.

```
def print_chorus():  
    print("Supercali...")  
    (etc)
```

```
def print_um_diddle():  
    print("Um diddle diddle...")  
    (etc)
```

```
def print_verse1():  
    print("Because I was afraid to speak...")  
    (etc)
```

```
# A function for the "main" program.
```

```
def main():  
    print_chorus() # Print the chorus  
    print_um_diddle() # Print the um diddles  
    print_verse1() # Print the 1st verse  
    print_chorus() # Print the chorus again  
    print_um_diddle() # Print the um diddles again  
    print_verse2() # Print the 2nd verse  
    print_chorus() # Print the chorus the last time
```

```
main() # Start the program
```

Function definitions

Function calls

- When a function is called, Python will
  - "jump" to the first line of the function's definition,
  - run all the lines of code inside the definition, then
  - "jump" back to the point where the function was called.

- You are in charge of dessert for Thanksgiving dinner. You decide to make two pumpkin pies and an apple pie.
- Write a program that ***defines*** three functions:
  - `make_apple()` should print a description of how to make an apple pie
  - `make_pumpkin()` should print a description of how to make a pumpkin pie
  - `main()` should ***call*** `make_apple()` and `make_pumpkin()` appropriately to make the pies.
- Don't forget to call `main()` at the end of your code!

- You want to write a program to sing (ok, print) the song "Happy Birthday" to you and your twin sibling.
- Here's how you might have done it before you learned about functions.

- Re-write this program to use functions.
- Define a function called `sing_song` that:
  - Asks the user to type in a name from the keyboard.
  - Sings the happy birthday song using that name.
- Define a `main ( )` function that calls your `sing_song ( )` function twice.
- It's OK that the program doesn't distinguish between your name and your twin's name.
  - That is, the program will use the same language to prompt the user for both names.

- What if we want the program to work exactly like the first program did, so it asks for my name and then my twin's name using different language?

```
def sing_song():
    print("Happy bday to you, happy bday to you!")
    print("Happy bday dear", name, "happy bday to you")

def main():
    name = input("What is your name? ")
    sing_song()
    twin_name = input("What is your twin's name? ")
    sing_song()

main()
```

This program doesn't  
work!



```
def sing_song():
    print("Happy bday to you, happy bday to you!")
    print("Happy bday dear", name, "happy bday to you")

def main():
    name = input("What is your name? ")
    sing_song()
    name = input("What is your twin's name? ")
    sing_song()

main()
```

This program doesn't  
work either!

# Local variables

- Every variable assigned to inside a function is "owned" by that function.
- These variables are *invisible* to all other functions.
- These are called *local variables* because they can only be used "locally" (within their own function).

# THIS PROGRAM DOESN'T WORK!

```
def sing_song():  
    print("Happy bday to you, happy bday to you!")  
    print("Happy bday dear", name, "happy bday to you")  
  
def main():  
    name = input("What is your name? ")  
    sing_song()  
    name = input("What is your twin's name? ")  
    sing_song()  
  
main()
```

Attempting to use `name` here will cause an error.

`name` is a local variable – it is invisible to all functions except `main()`.



- This is a problem.
- We'd like some way for functions to communicate with each other.
- Specifically, we'd like a way for main to send the value of the variable name to `sing_song` so `sing_song` may use it.

- `main()` needs to send the variable name to `sing_song`.
- Step 1: add name inside the parentheses in `sing_song`'s definition.
  - This tells `sing_song` that the value of this variable will come from the function that calls `sing_song`.

## BEFORE

```
def sing_song():  
    print("Happy bday to you, happy bday to you!")  
    print("Happy bday dear", name, "happy bday to you")
```

## AFTER

```
def sing_song(name):  
    print("Happy bday to you, happy bday to you!")  
    print("Happy bday dear", name, "happy bday to you")
```

- `main()` needs to send the variable name to `sing_song`.
- Step 1: add name inside the parentheses in `sing_song`'s definition.
  - This tells `sing_song` that the value of this variable will come from the function that calls `sing_song`.
- Step 2: every place that `sing_song` is called, inside the parentheses of the call, put whatever value you want to send to `sing_song`.

## BEFORE

```
def main():  
    name = input("What is your name? ")  
    sing_song()  
    name = input("What is your twin's name? ")  
    sing_song()
```

## AFTER

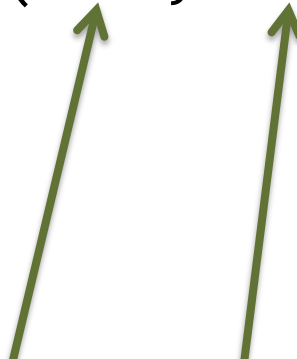
```
def main():  
    my_name = input("What is your name? ")  
    sing_song(my_name)  
    twin_name = input("What is your twin's name? ")  
    sing_song(twin_name)
```



# Arguments and parameters

Defining:

```
def name_of_function(var1, var2, ...):  
    statement  
    statement  
    statement
```



Calling:

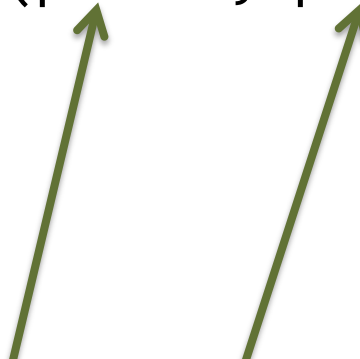
```
name_of_function(value1, value2, ...)
```

When a function is called, all the values inside the parentheses from the calling line are ***immediately copied*** into the variables given in the function definition.

# Arguments and parameters

Defining:

```
def name_of_function(param1, param2, ...):  
    statement  
    statement  
    statement
```



Calling:

```
name_of_function(arg1, arg2, ...)
```

The values being copied from the calling function are called ***arguments***.

The variables being copied into are called ***parameters***.

```
def sing_song(name):  
    print("Happy bday to you, happy bday to you!")  
    print("Happy bday dear", name, "happy bday to you")  
  
def main():  
    my_name = input("What is your name? ")  
    sing_song(my_name)  
    twin_name = input("What is your twin's name? ")  
    sing_song(twin_name)  
  
main()
```

```
def sing_song(name):  
    print("Happy bday to you, happy bday to you!")  
    print("Happy bday dear", name, "happy bday to you")  
  
def main():  
    my_name = input("What is your name? ")  
    sing_song(my_name)  
    twin_name = input("What is your twin's name? ")  
    sing_song(twin_name)  
  
main()
```

When Python runs the red line, it copies the value of `my_name` into `sing_song`'s variable name.

```
def sing_song(name):
    print("Happy bday to you, happy bday to you!")
    print("Happy bday dear", name, "happy bday to you")

def main():
    my_name = input("What is your name? ")
    sing_song(my_name)
    twin_name = input("What is your twin's name? ")
    sing_song(twin_name)

main()
```

When Python runs the blue line, it copies the value of `twin_name` into `sing_song`'s variable name.

```
def sing_song(name):  
    print("Happy bday to you, happy bday to you!")  
    print("Happy bday dear", name, "happy bday to you")
```

```
def main():  
    name = input("What is your name? ")  
    sing_song(name)  
    name = input("What is your twin's name? ")  
    sing_song(name)
```

```
main()
```

- You *may* use the same variable names in both places, if desired.
- Each function then has its own copy of the variable.
- There is no permanent link between the variables.

```
def some_function(x):  
    print("Inside the function, x is", x)  
    x = 17  
    print("Inside the function, x is changed to", x)  
  
def main():  
    x = 2  
    print("Before the function call, x is", x)  
    some_function(x)  
    print("After the function call, x is", x)
```

```
main()
```

**Output:**

```
Before the function call, x is 2  
Inside the function, x is 2  
Inside the function, x is 17  
After the function call, x is 2
```

# Wait. What?

- There is no permanent connection between the `x` in `main` and the `x` in `some_function`.
- Arguments are passed --- one way only --- from `main` to `some_function` when `main` calls `some_function`.
  - This copies `main`'s value of `x` into `some_function`'s `x`.
- Any assignments to `x` inside of `some_function` do not come back to `main`.



- *"That sounds like local variables."*
- Yes, just as local variables are invisible outside of the functions that own them, variables used as parameters inside a function definition are local to that function.
- So parameters in a function definition are really local variables that have values automatically copied into them when the function is called.

# You've seen arguments already

- `name = input("What is your name? ")`
- `x = 5`
- `y = 2`
- `print("x is", x, "y is", y)`
- `print("their sum is", x + y)`

Arguments can be variables, literals, or math expressions.

- You no longer have a twin. Now you have a sibling that is two years older than you, but you still share the same birthday.
- Edit birthday4.py so sing\_song now will print the lyrics ***but also print how old the person is.***
- Add a second parameter to sing\_song called age.
- Edit main() to ask for your age, as well as your name and sibling's name.
- Edit the two calls to sing\_song so appropriate ages are passed as arguments.