

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.

Function definitions

```
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
```

Function calls

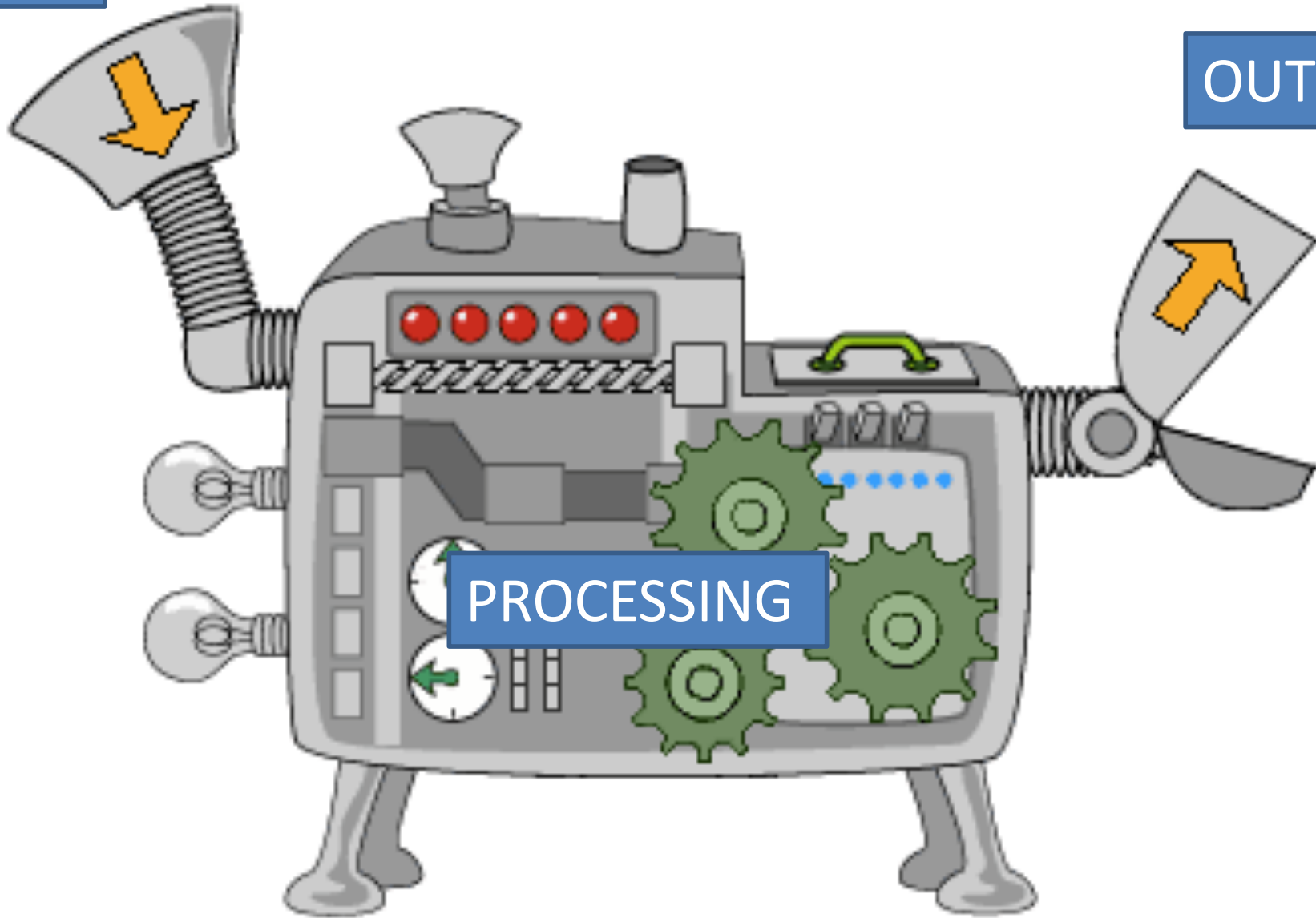
```
main() # Start the program
```

- 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.

- 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.

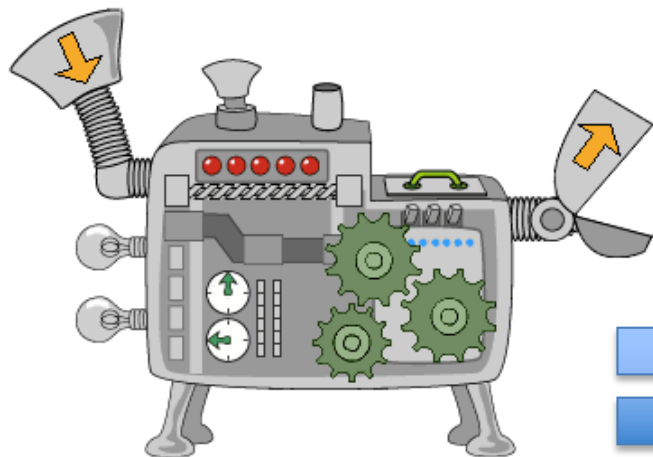
```
1  def twinkle():  
2      print("Twinkle twinkle little star")  
3      print("How I wonder what you are")  
  
4  def main():  
5      twinkle()          # Call (run) the twinkle function.  
6      print("Up above the world so high")  
7      print("Like a diamond in the sky")  
8      twinkle()          # Call the twinkle function again.  
  
9  main()                # Call main() to start the program.
```

INPUT

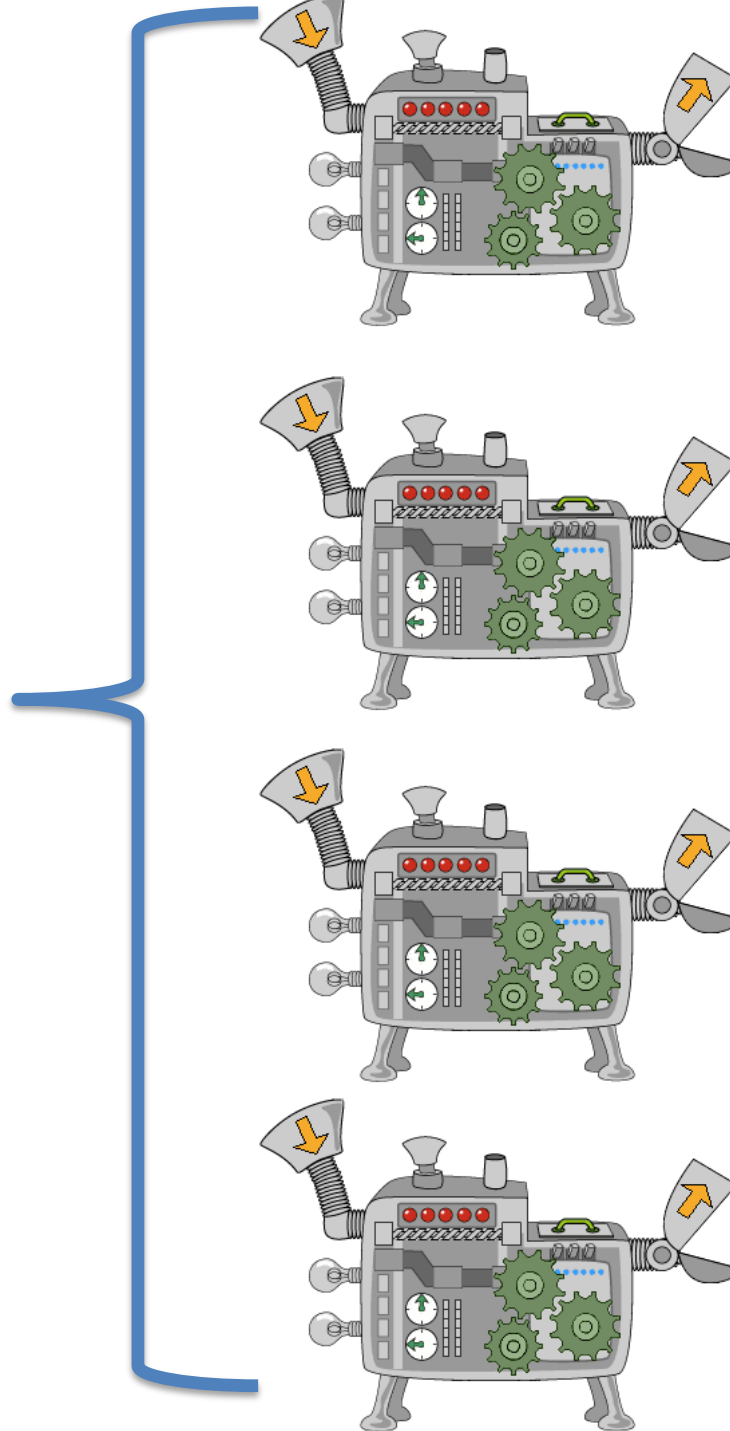
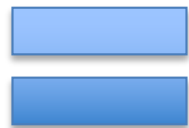


OUTPUT

PROCESSING



Make dinner



Make appetizer

Make salad

Make entree

Make dessert

- Suppose we want to write a program to sing "Happy Birthday."
- If we think of "sing Happy Birthday" as an algorithm, what information does the algorithm require as input?

I could write a function like this:

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

This is usually not the best way:

- Forces the user to use the same input prompt every time.
- Doesn't let the person's name come from another input source other than the keyboard.

Arguments and Parameters

- Algorithms described by functions allow for *input* via arguments and parameters.
- This method allows you to send information into a function to change its behavior when it runs.
- More flexible than using input statements at the beginning of a function.

Arguments and parameters

Defining:

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

- ***Parameters*** are variables placed inside the parentheses when a function is ***defined***.
- They should represent pieces of information that the function needs to know ***ahead of time in order to run***.


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

- The statements inside a function definition can use the parameters as normal variables.
- Notice how the parameters aren't defined inside the function. (There is no variable assignment statement like `name = something`). The value of the parameter must come from outside the function.

Arguments and parameters

Defining:

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

Two green arrows originate from the arguments 'arg1' and 'arg2' in the function call below and point upwards to the parameters 'param1' and 'param2' in the function definition above, illustrating the mapping of arguments to parameters.

Calling:

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

The values placed inside the parentheses when a function is called are known as ***arguments***.

They provide the extra information that the function needs to do its job.

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.

(Anything you could put on the right side of a variable assignment statement can be an argument.)

Determining good parameters

- Suppose you are writing a function to compute the area of a rectangle. What outside information does the function need to be given to work?
- What if you're writing a function to determine a person's age. What outside information would this function need?

- Suppose I have an identical twin. I want my program to ask for our names and then sing Happy Birthday to both of us, individually.



```
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.

Local variables

- Any variable used as a parameter inside a function is "owned" by that function, and is *invisible* to all other functions.
- These are called *local variables* because they can only be used "locally" (within their own function).
- Any variable created inside a function is also a local variable and cannot be seen outside of that function.

```
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`.

- 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 `birthday.py` so `sing_song` now will print the lyrics ***and also print how old the person is.***
- Add a second parameter to `sing_song` called `age` (representing the age the person is turning), and add a third print statement within the function to display a message with the new age.
- Edit `main()` to ask for your age, as well as your name and sibling's name. Don't ask for your sibling's age using `input()`, since you know they're always two years older than you.
- Edit the two calls to `sing_song` so appropriate ages are passed as arguments.
- Challenge: write a function that takes three integer arguments (month, day, year) and prints if the year is a leap year and the day of the week (Monday, Tuesday, etc) that the date corresponds to.