Unit II

Functions and Strings

```
Int Abs(int n)
{
        If (n<0)
        {
                N=-n;
                Return n;
        }
        Else
        {
                N=n;
                Return n;
        }
}
Abs(-5);
Abs(5);
```

Function

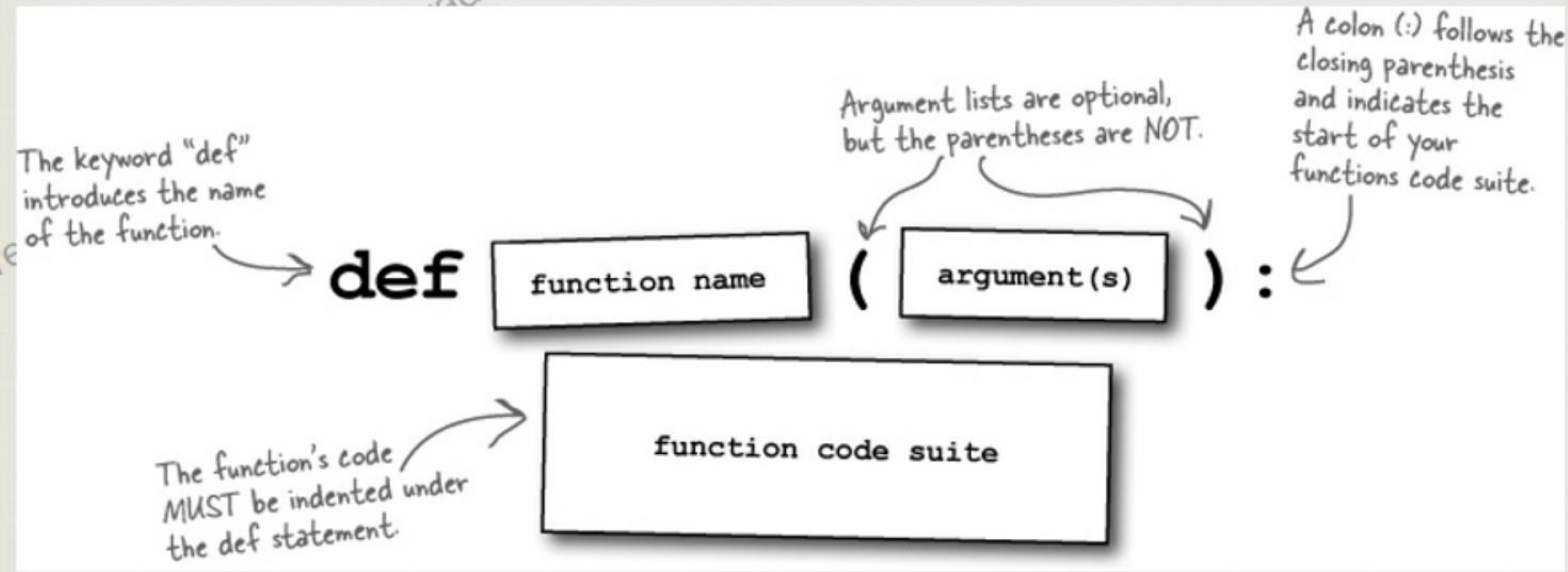
- In Python, function is a group of related statements that perform a specific task.
- Function helps to break our program into smaller and modular chunks.
- As our program grows larger and larger, functions make it more organized and manageable.
- Furthermore, it avoids repetition and makes code reusable.
- In the context of programming, a **function** is a named sequence of statements that performs a computation.

TYPES

- There are two kinds of functions in Python.
- Built-in functions that are provided as part of Python - input(), type(), float(), int() ...
- Functions that we define ourselves and then use

Syntax of Function


```
def function_name(parameters):
"""docstring"""
    statement(s)
```



- Keyword def marks the start of function header.
- A function name to uniquely identify it.
- Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of function header.
- Optional documentation string (docstring) to describe what the function does.
- One or more valid python statements that make up the function body.
- Statements must have same indentation level (usually 4 spaces).
- An optional return statement to return a value from the function.

```
big = max([5,8,2])
```

Argument

```
big = max('hello world')
```

Assignment

'w' Result

Function calls

- >>> type(32)
- <type 'int'>

Type conversion functions

- built-in functions

Type Conversions

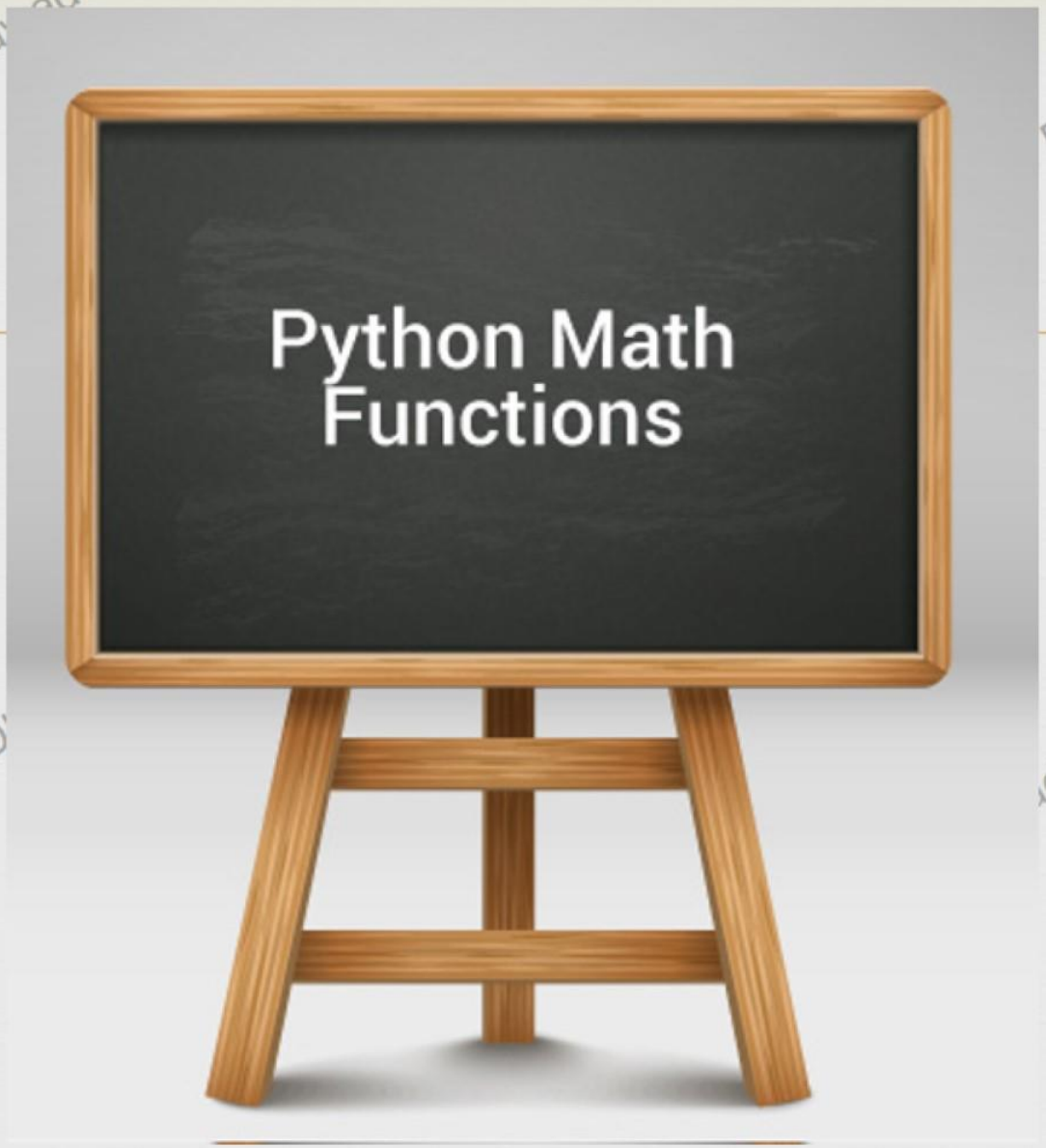
- When you put an integer and floating point in an expression the integer is implicitly converted to a float
- You can control this with the built in functions int() and float()

```
>>> print float(99) / 100
0.99
>>> i = 42
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print f
42.0
>>> type(f)
<type 'float'>
>>> print 1 + 2 * float(3) / 4 - 5
-2.5
```

String Conversions

- You can also use int() and float() to convert between strings and integers
- You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<type 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str'
and 'int'
>>> ival = int(sval)
>>> type(ival)
<type 'int'>
>>> print ival + 1
124
>>> nsv = 'hello everyone'
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```



- Python has a math module that provides most of the familiar mathematical functions.
- A **module** is a file that contains a collection of related functions.

Function	Returns (description)
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x
cmp(x, y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: e^x
fabs(x)	The absolute value of x.
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0.
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity

min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
pow(x, y)	The value of x**y.
round(x [,n])	x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.

```
>>> ratio = signal_power / noise_power
>>> decibels = 10 * math.log10(ratio)

>>> radians = 0.7
>>> height = math.sin(radians)

>>> degrees = 45
>>> radians = degrees / 360.0 * 2 * math.pi

>>> math.sin(radians)
>>> math.sqrt(2) / 2.0
```

Composition

- Python functions can be **composed**, meaning that you use the result of one function as the input to another.
- >>>x=2
- >>> y = math.exp(math.log(x+1))
- >>> y
- 3.0000000000000004

```
def print_twice(n):
        print(n,n)
>>> print_twice("Hello")
Hello Hello
>>> print_twice(2)
2 2
>>> print_twice(abs(-2))
2 2
>>> print_twice(max(3,4,abs(-5),6))
6 6
>>>
```

Adding New Functions

- A **function definition** specifies the name of a new function and the sequence of statements that execute when the function is called.
- We create a new function using the def keyword followed by optional parameters in parenthesis.
- We indent the body of the function.
- This defines the function but *does not* execute the body of the function

```
def print_lyrics():
    print("I'm a jack, and I'm okay.")
    print ("I sleep all night and I work all day.")
```

```
def print_lyrics():
        print("Hi")
def repeat_lyrics():
        print_lyrics()
        print_lyrics()
```

Flow of execution

- The order in which statements are executed, which is called the **flow of execution**.
- Execution always begins at the first statement of the program.
- Statements are executed one at a time, in order from top to bottom.
- Function definitions do not alter the flow of execution of the program
- But statements inside the function are not executed until the function is called.

- A function call is like a detour in the flow of execution.
- Instead of going to the next statement, the flow jumps to the body of the function, executes all the statements there, and then comes back to pick up where it left off.

Definitions and Uses

- Once we have defined a function, we can call (or invoke) it as many times as we like
- This is the store and reuse pattern

```
x = 5
print('Hello‘)

def print_lyrics():
    print("I'm a jack, and I'm okay.”)
    print (“I sleep all night and I work all day.”)

print('Yo‘)
print_lyrics()
x = x + 2
print(x)
```

O/P:
Hello
Yo
I'm a jack, and I'm okay.
I sleep all night and I work all day.
7

Parameters and arguments

- Inside the function, the arguments are assigned to variables called **parameters**

Arguments

- An argument is a value we pass into the function as its input when we call the function
- We use arguments so we can direct the function to do different kinds of work when we call it at different times
- We put the arguments in parenthesis after the name of the function

```
big = max('Hello world')
```

Argument

Parameters

- A parameter is a variable which we use in the function definition that is a "handle" that allows the code in the function to access the arguments for a particular function invocation.

```
def greet(lang):
        if lang == 'es':
                print('Hola')
        elif lang == 'fr':
                print('Bonjour')
        else:
                print('Hello')
>>> greet('en')
Hello
>>> greet('es')
Hola
>>> greet('fr')
Bonjour
>>>
```

Variables and Parameters Are Local

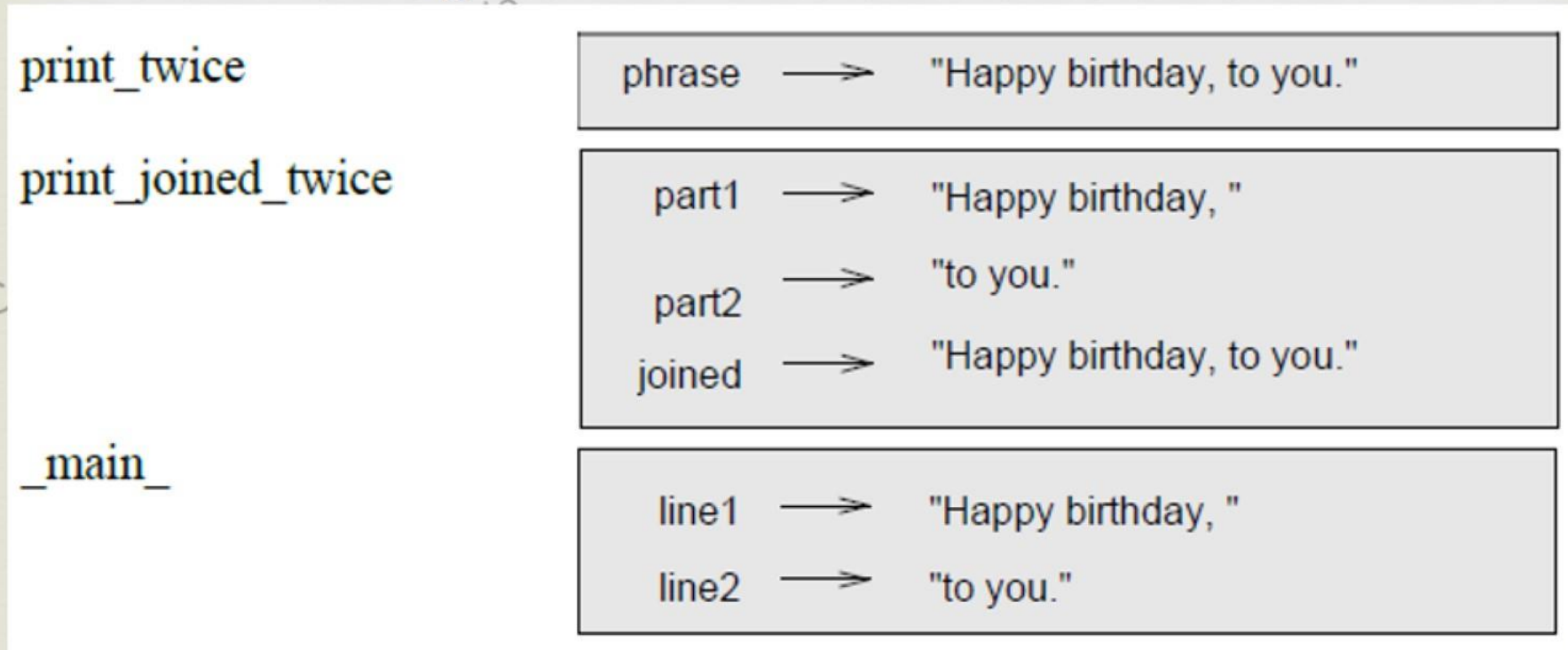
```
def print_twice(phrase):
    print (phrase)
    print (phrase)
```

```
def print_joined_twice(part1, part2) :
        joined = part1 + part2
        print_twice(joined)
>>> line1 ="Happy birthday,"
>>> line2 = "to you."
>>> print_joined_twice(line1, line2)
>>>print(joined)
```

NameError: name 'joined' is not defined

Stack Diagrams

- Like state diagrams, stack diagrams show the value of each variable, but they also show the function each variable belongs to.



Main( )
Print_joined_twice()
Print_twice()

```
Traacebck (innermost last):
File "test.py", line 13, in __main__
print_joined_twice(line1, line2)
File "test.py", line 5, in print_joined_twice
print_twice(joined)
File "test.py", line 9, in print_twice
print(joined)
NameError: name ‘joined' is not defined
```

Fruitful functions and void functions

- x = math.cos(radians)
- golden = (math.sqrt(5) + 1) / 2

Why Functions?

- Creating a new function gives you an **opportunity to name a group of statements**,which makes your program easier to read and debug.
- Functions can make a program smaller **by eliminating repetitive code**. Later, if you make a change, you only have to make it in one place.
- Dividing a long program into functions allows you to **debug the parts one at a time** and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, you can **reuse** it.

```
def factorial (n):
    if not isinstance(n, int):
        print 'Factorial is only defined for integers.'
        return None
    elif n < 0:
        print 'Factorial is not defined for negative integers.'
        return None
    elif n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Strings

- **A string is a sequence**
- >>> fruit = ‘apple'
- >>> letter = fruit[1]

- >>> letter = fruit[1.5]
- TypeError: string indices must be integers, not float

Traversal with a for loop

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print letter
    index = index + 1

for char in fruit:
    print char
```

- prefixes = 'JKLMNOPQ'
- suffix = 'ack'
- for letter in prefixes:
- print letter + suffix

String slices

- >>> s = 'Monty Python'
- >>> print s[0:5]
- Monty
- >>> print s[6:12]
- Python

Strings are immutable

- >>> greeting = 'Hello, world!'
- >>> greeting[0] = 'J'
- TypeError: 'str' object does not support item assignment

- >>> greeting = 'Hello, world!'
- >>> new_greeting = 'J' + greeting[1:]
- >>> print new_greeting

Searching

```
def find(word, letter):
    index = 0
    while index < len(word):
        if word[index] == letter:
            return index
        index = index + 1
    return -1
```

Exercise 8.4. Modify find so that it has a third parameter, the index in word where it should start looking.

Looping and counting

```
word = ‘apple'
count = 0
for letter in word:
        if letter == ‘p':
                count = count + 1
print(count)
```

String methods

- Lower()
- Upper()
- Find()
- Replace()
- Len()

The in operator

- The word in is a boolean operator that takes two strings and returns True if the first appears as a substring in the second
- >>> 'a' in 'banana'
- True
- >>> 'seed' in 'banana'
- False

```
def in_both(word1, word2):
        for letter in word1:
                if letter in word2:
                        print letter
```

String comparison

```
if word < ‘kiwi':
        print('Your word,' + word + ', comes before kiwi.‘)
elif word > ‘kiwi':
        print('Your word,' + word + ', comes after kiwi.‘)
else:
        print('All right, kiwis.‘)
```