Functions
Built-in Functions

- Like miniature programs
  - Receive input
  - Process the input
  - Have output
- Some Python built-in functions

Function	Example	Input	Output
int	int(2.6) is 2	number	number
chr	chr(65) is “A”	number	string
ord	ord('A') is 65	string	number
round	round(2.34, 1) is 2.3	number, number	number

Built-in Functions

- Output of functions is a single value
  - Function is said to return its output
- Items inside parentheses called arguments
- Examples:

```
num = int(3.7)          # literal as an argument

num1 = 2.6
num2 = int(num1)        # variable as an argument

num1 = 1.3
num2 = int(2 * num1)    # expression as an argument
```

User-defined Functions

- Defined by statements of the form

```
def functionName(par1, par2, ...):
    indented block of statements()
    return expression
```

  - par1, par2 are variables (called parameters)
  - Expression evaluates to a literal of any type
  - Header must end with colon
  - Each statement in block indented same

User-defined Functions

- Passing parameters
  - We consider here pass by position
  - Arguments in calling statement matched to the parameters in function header based on order
- Parameters and return statements optional in function definitions
- Function names should describe the role performed

Functions Having One Parameter

```
def fahrenheitToCelsius(t):
    ## Convert Fahrenheit temperature to Celsius.
    convertedTemperature = (5 / 9) * (t - 32)
    return convertedTemperature

def firstName(fullName):
    ## Extract the first name from a full name.
    firstSpace = fullName.index(" ")
    givenName = fullName[:firstSpace]
    return givenName
```

keyword signifying function definition

function name

parameter

```
def fahrenheitToCelsius(t):
```

Functions Having One Parameter

- Example: Program uses the function *fahrenheitToCelsius*

```
def fahrenheitToCelsius(t):
    ## Convert Fahrenheit temperature to Celsius.
    convertedTemperature = (5 / 9) * (t - 32)
    return convertedTemperature

fahrenheitTemp = eval(input("Enter a temperature in degrees Fahrenheit: "))
celsiusTemp = fahrenheitToCelsius(fahrenheitTemp)
print("Celsius equivalent:", celsiusTemp, "degrees")
```

[Run]

```
Enter a temperature in degrees Fahrenheit: 212
Celsius equivalent: 100.0 degrees
```

Passing a Value to a Function

- If the argument in a function call is a variable
  - Object pointed to by the argument variable (not the argument variable itself) passed to a parameter variable
  - Object is immutable, there is no possibility that value of the argument variable will be changed by a function call

Passing a Value to a Function

- Example: Program shows there is no change in the value of the argument

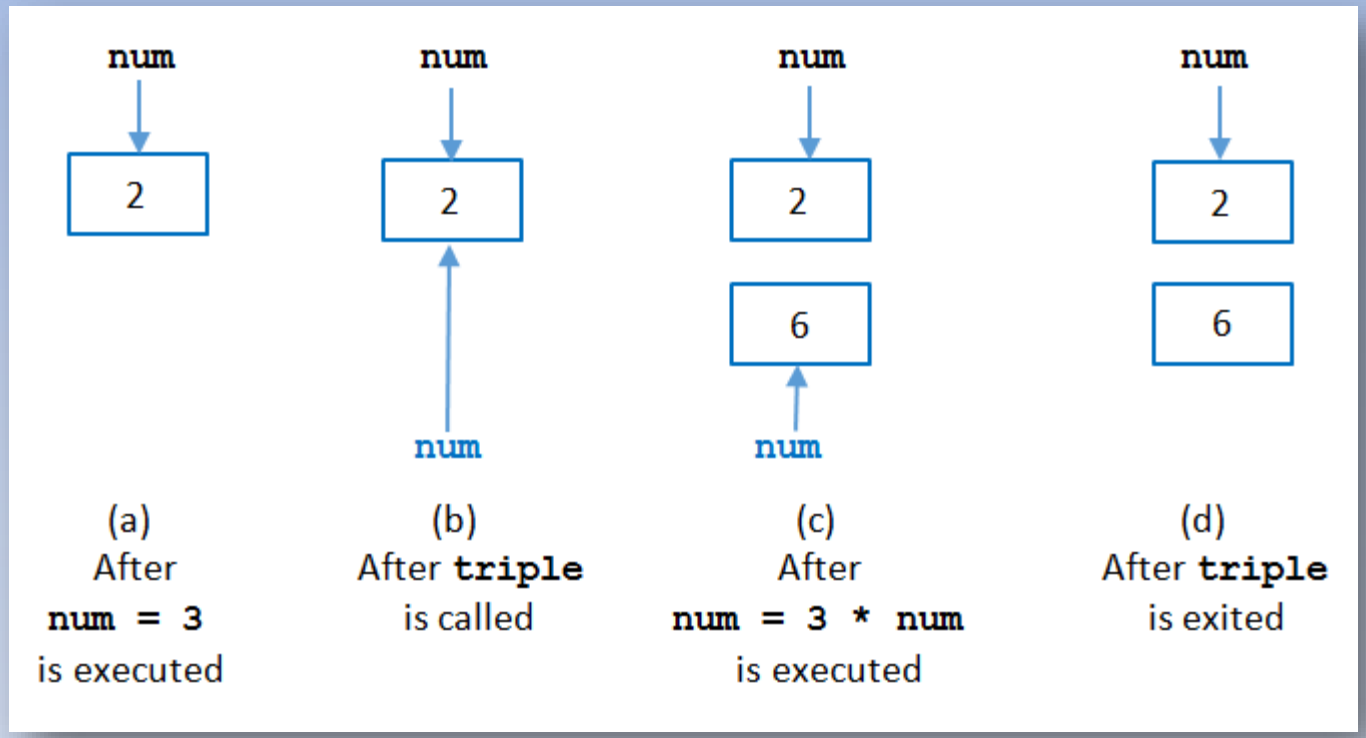
```
def triple(num):
    num = 3 * num
    return num

num = 2
print(triple(num))
print(num)
```

[Run]

```
6
2
```

Passing a Value to a Function



Passing a value to a function

Functions Having Several Parameters

- Must be the same number of arguments as parameters in the function
- Data types of arguments' values must be compatible with data types expected by the parameters
  - Must also be in the same order

Functions Having Several Parameters

- Example: Program uses the function *pay*.

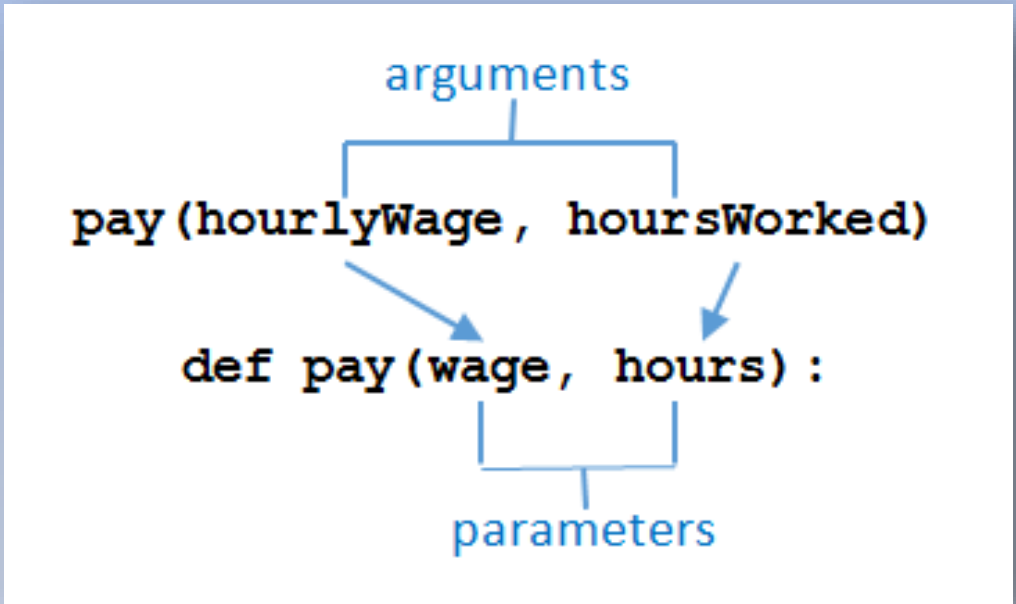
```
def pay(wage, hours):
    ## Calculate weekly pay with time-and-a-half for overtime.
    if hours <= 40:
        amount = wage * hours
    else:
        amount = (wage * 40) + ((1.5) * wage * (hours - 40))
    return amount

## Calculate a person's weekly pay.
hourlyWage = eval(input("Enter the hourly wage: "))
hoursworked = eval(input("Enter the number of hours worked: "))
print("Earnings: ${0:,.2f}".format(pay(hourlyWage, hoursWorked)))
```

[Run]

```
Enter the hourly wage: 24.50
Enter the number of hours worked: 45
Earnings: $1,163.75
```

Functions Having Several Parameters



Passing arguments to a function

Python Program Execution with User-defined Functions

- Concept of a “main” function
- Program execution starts in the __main__ module
- Single module Vs Multiple modules

```
#Program execution with User-defined Functions

def fahrenheitToCelsius(t):
    ## Convert Fahrenheit temperature to Celsius.
    convertedTemperature = (5 / 9) * (t -32)
    return convertedTemperature

#Below is the "main" function which acts as the entry point
if __name__ == '__main__':
    fahrenheitTemp = eval(input("Enter a temperature in degrees Fahrenheit: "))
    celsiusTemp = fahrenheitToCelsius(fahrenheitTemp)
    print("Celsius equivalent:", round(celsiusTemp,2), "degrees")
```

Functions calling other functions

- A function can call another function
- When the called function terminates
  - Control returns to the place in a calling function just after where the function call occurred

```python
def function1(arg1):
    # ............
    result1 = function2(arg1, arg1+250)
    return(result1)

def function2(arg2, arg3):
    # ............
    result2 = arg2 + arg3
    return(result2)

# notion of a 'main' function
# control the program flow in this part
if __name__ == "__main__":
    result = function1(500)
    print("Result:", result)
```

```python
def function1(arg1):
    # ............
    result1 = function2(arg1, arg1+250)
    return(result1)

def function2(arg2, arg3):
    # ............
    result2 = arg2 + arg3
    return(result2)

# notion of a 'main' function
# control the program flow in this part
def main():
    result = function1(500)
    print("Result:", result)

main()
```

Functions returning Multiple values

- Functions can return any type of object
- For example, let's say you pass two numeric values to a function (via arguments) and want the function to return their sum AND their product

```
def sum_and_product(x1, x2):
    result_sum = x1 + x2
    result_product = x1 * x2
    return(result_sum, result_product)

def main():
    number1 = 1234
    number2 = 5678
    s, p = sum_and_product(number1, number2)
    print("sum =", s, "product =", p)

main()
```

Case study writing a function for *Factorial*

- From probability theory if we want to compute the number of different ways in which “r” items can be selected from a collection of “n” items without regard to order:

$$C(n, r) = \frac{n!}{r!\,(n-r)!}$$

- As an example if n=5 and r=2

$$C(5, 2) = 5! / (2! * (5-2)!) = 10$$

- *In Mathematics we use more precise language:*
  - *When the order doesn't matter, it is a **Combination,** when the order **does** matter it is a **Permutation**.*

Scope of Variables

- Variable created inside a function can only be accessed by statements inside that function
  - Ceases to exist when the function is exited
- Variable is said to be local to function or to have local scope
- If variables created in two different functions have the same name
  - They have no relationship to each other

Scope of Variables

- Example: Variable x in the function main, variable x in the function trivial are different variables

```
def main():
    ## Demonstrate the scope of variables.
    x = 2
    print(str(x) + ": function main")
    trivial()
    print(str(x) + ": function main")

def trivial():
    x = 3
    print(str(x) + ": function trivial")

main()
```

[Run]

```
2: function main
3: function trivial
2: function main
```

Scope of Variables

- Example: Variable x created in function main not recognized by function trivial.

```
def main():
    ## Demonstrate the scope of local variables.
    x = 5
    trivial()

def trivial():
    print(x)

main()
```

Scope of Variables

- Scope of a variable is the portion of the program that can refer to it
- To make a variable global, place assignment statement that creates it at top of program.
  - Any function can read the value of a global variable
  - Value cannot be altered inside a function unless

```
global globalVariableName
```

Scope of Variables

- Example: Program contains a global variable

```
x = 0   # Declare a global variable.

def main():
    ## Demonstrate the scope of a global variable.
    print(str(x) + ": function main")
    trivial()
    print(str(x) + ": function main")

def trivial():
    global x
    x += 7
    print(str(x) + ": function trivial")

main()
```

[Run]

```
0: function main
7: function trivial
7: function main
```

Functions that do not Return Values

- Example: Program displays three verses of children's song

```
def oldMcDonald(animal, sound):
    print("Old McDonald had a farm. Eyi eyi oh.")
    print("And on his farm he had a", animal + ".", "Eyi eyi oh.")
    print("With a", sound, sound, "here, and a", sound, sound, "there.")
    print("Here a", sound + ",", "there a", sound + ",", \
          "everywhere a", sound, sound + ".")
    print("Old McDonald had a farm. Eyi eyi oh.")

## Old McDonald Had a Farm
oldMcDonald("lamb", "baa")
print()
oldMcDonald("duck", "quack")
print()
oldMcDonald("cow", "moo")
```

[Run]

```
Old McDonald had a farm. Eyi eyi oh.
And on his farm he had a lamb. Eyi eyi oh.
With a baa baa here, and a baa baa there.      ... etc. ...
```

Functions without Parameters

- Example: Program calculates the population density of a state

```
def main():
    ## Calculate the population density of Hawaii.
    describeTask()
    calculateDensity("Hawaii", 1375000, 6423)

def describeTask():
    print("This program displays the population")
    print("density of the last state to become")
    print("part of the United States.\n")

def calculateDensity(state, pop, landArea):
    density = pop / landArea
    print("The density of", state, "is")
    print("{0:,.2f} people per square mile.".format(density))

main()
```

[Run]

```
This program displays the population
density of the last state to become
part of the United States.

The density of Hawaii is
214.07 people per square mile.
```

Library Modules

- A library module is a file with extension *.py*
  - Contains functions and variables
  - Can be used (imported) by any program
  - can be created in Spyder or IDLE or any text editor
  - Looks like an ordinary Python program
- To gain access to the functions and variables
  - place a statement of the form *import moduleName* at the beginning of the program

Library Modules

- Consider a file with *pay* and *futureValue* functions
  - Save as *finance.py* in same folder as the program that's going to call these functions
  - Use the import statement to include these functions

```
import finance.py

## Calculate a person's weekly pay.
hourlyWage = eval(input("Enter the hourly wage: "))
hoursworked = eval(input("Enter the number of hours worked: "))
print("Earnings: ${0:,.2f}".format(finance.pay(hourlyWage, hoursWorked)))
```

Library Modules

Module	Some Tasks Performed by its Functions
os	delete and rename files.
os.path	determine whether a file exists in a specified folder. This module is a submodule of os.
pickle	store objects (such as dictionaries, lists, and sets) in files and retrieve them from files.
random	randomly select numbers and subsets.
tkinter	enable programs to have a graphical user interface.
turtle	enable turtle graphics.

Several modules from the standard library