

Functions and Dictionary

User-defined function


- Define a function named `area_rectangle()` to accept two variables `side1` and `side2` and computes area and returns it

```
def area_rectangle(side1,side2):  
    area=side1*side2  
    return area
```

Two arguments are passed to function and function returns value stored in variable `area`

Default argument value

Default argument : Should be last ones in the argument list



```
def welcome(name, msg = "Good morning!"):
    """
    This function welcomes the person with the provided
    message. If message is not given, then it defaults to
    "Good morning!"
    """
    print ("Hello",name + ', ' + msg)

welcome("All") #One argument
welcome("Everyone","How do you do?") # 2 arguments
i.e. default is not used
```

Output:

```
Hello All, Good morning!
Hello Everyone, How do you do?
```

help(welcome) gives multiline comment written after function header

Keyword argument

- Sometimes, a function has lots of parameters and most of them are default, then it is not necessary to call remaining arguments in the same order as apperaing in the definition, but we can call them with their name
- Such arguments call using name are keyword/name arguments
- Else, at the calling point these are referred as positional arguments

<pre>def abc (a, b, c = 1): return a * b + c print(abc (1, 2)) print (abc(1, 2, 3)) print abc (c = 5, b = 2, a = 2) print abc (b = 2, a = 2) print abc (6, c = 2, b = 1) print abc(8, b = 2)</pre>	<p>Function abc() has three arguments with one as default argument</p> <p>Function abc() is called with two positional and uses one default argument, # returns 3</p> <p>Function abc() is called with three positional arguments, #returns 5</p> <p>Function abc() is called with three named arguments, #returns 9, order not matters</p> <p>Function abc() is called with two named arguments, #and uses one default arguments, returns 5</p> <p>Fn uses one positional and two named arguments, returns 8</p> <p>Fn uses one positional, one named and one default argument, returns 17</p>
--	---

Importing user-defined module for reusability of code and to have modular program

- Store the function in module say area.py
- Include path in the file where the defined function is to be used e.g.

```
import sys
sys.path.append('/home/sharanjit/Desktop/python-pscs-sem1/python-program')
import area
```

- Use the defined function `area_rectangle()` in area module as `modulename.fnname()` e.g.

```
area.area_rectangle(breadth,width)
```

Example: Importing function

Area.py

```
def area_figure(side1,side2):  
    '''function to find area of rectangle'''  
    typefig='square' if side1==side2 else 'rectangle'  
    area=side1*side2  
    return (typefig,area)
```

Main file

```
import sys  
import os  
os.system('clear')  
#sys.path.append('/home/sharanjit/Desktop/python-pscs-  
seml/python-program')  
import area  
help(area.area_figure)  
T1= area.area_figure(5,10)  
print("given figure is ", T1[0], and "area= ",T1[1])
```

Ternary operator:

value = true-expr if condition else false-expr

Usage: Condense code

Limitation: at cost of readability in case the condition as well as the true and false expressions are very complex. |

More on tuple..

- Tuple is mutable, but can have element of mutable sequence
- `tup = tuple(['foo', [1, 2], True])`
- `tup[2] => error`
- `Tup[1].append(7) => ('foo', [1, 2, 7], True)`

DICTIONARY

- Dictionaries are unordered sets.
- Common *hash map* or *associative array*
- Main difference between list and dictionary
 - is that items in dictionaries are accessed via keys and not via their position
 - Does not support sequence operation as in like strings, tuples and lists
- Mutable, heterogeneous and Unordered key-value-pairs
- Use parentheses {key:value pairs} and values may be set/list/dictionary etc.
 - `d2={'subjects':('maths','eng','cs'),'mm':(100,50,100)}`
- **Examples:**
 - `weather_data={'Delhi':34, 'Mumbai':37, 'Patna': 38}; print(weather_data)`
 - => same elements but order may be different

Accessing elements in dictionary

- Use keys to access value of elements as

`weather_data['Delhi'] => 34`

- Cannot use index value as in list
- Adding an new key-value pair
- `weather_data['jaipur']=40`
- Note that key and value need not to be of different types, either may be numeric or string
- `data={1:34, 'Bhutan':25, 'Mumbai':37, 'Patna': 38}` #heterogeneous elements
- Creating an empty dictionary as `dict1={}` or `dict1=dict()`
- Each key's value may be either of scalar/sequence/unmutable
- `Hash()` gives whether value can be used as a hash value/key value

ZIP()

- Given two lists as fruit=['mango','apple']
- price=[100,120]
- Dictionary is fruitprice={'mango':100,'apple':120}
- Assigning to other variable f=fruitprice
- From two lists having scalar values, a dictionary may be created after applying zip method on list
- Zip method returns a function returns a zip object, which is an iterator of tuples
- If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

Output??

- dishes = ["pizza", "sauerkraut", "paella", "hamburger"]
- countries = ["Italy", "Germany", "Spain", "USA"]
- country_specialities_iterator = zip(countries, dishes)
- l1=list(country_specialities_iterator)

list output is:

l1 => [('Italy', 'pizza'), ('Germany', 'sauerkraut'), ('Spain', 'paella'), ('USA', 'hamburger')]

dict1=dict(l1) => output??

{'Italy': 'pizza', 'Germany': 'sauerkraut', 'Spain': 'paella', 'USA': 'hamburger'}

Dict2=dict(country_specialities_iterator)

Same output as that of dict1

Operators

- **Operator** **Explanation**
- `len(d)` returns the number of stored entries, number of (key,value) pairs
- `del d[k]` deletes the key k together with his value
- `k in d` True, if a key k exists in the dictionary d
- `k not in d` True, if a key k doesn't exist in the **dictionary d**

Built-in functions/operator for Dictionary

Weekdays={1:'mon',2: 'tues',3:'wed',4:'thurs',5:'fri'}

Function name	Usage	output
keys():all keys	Weekdays.keys()	dict_keys([1, 2, 3, 4, 5])
values(): all values	Weekdays.values()	dict_values(['mon', 'tues', 'wed', 'thurs', 'fri'])
items(): all elements	Weekdays.items()	dict_items([(1, 'mon'), (2, 'tues'), (3, 'wed'), (4, 'thurs'), (5, 'fri')])
Len: length of dictionary	len(Weekdays)	5
in: membership	1 in Weekdays	True
Copy(): to get copy of weekdays just like list	To get a copy of dictionary so that change in copied dictionary is not reflected in original	W2=Weekdays.copy()

Weekdays={1:'mon',2: 'tues',3:'wed',4:'thurs',5:'fri'}

Function name	Usage	output
Max: maximum key	max(Weekdays)	5
Min: minimum key	min(Weekdays)	1
Sum: total of keys if keys are numeric else exception	max(Weekdays.values()) sum(weekdays.keys())	Wed 15
del: to remove an element	del Weekdays[1]	{2: 'tues', 3: 'wed', 4: 'thurs', 5: 'fri'}
Del dictionaryname	2 in Weekdays 'xxx' in weekdays	True False
clear()	Weekdays.clear()	Empty dict
get(key,return value incase not found)	Weekdays.get(3,-1) Weekdays.get(7,-1)	Wed -1
Update(): To add another dictionary/element	Weekdays.update(more) More={6:'sat',7:'sun'}	Weekdays={1:'mon',2: 'tues',...,7:'sun'}

Deleting elements

- `W=Weekdays={1:'mon',2: 'tues',3:'wed',4:'thurs',5:'fri'}`

Popitem(): To remove any arbitrary pair	<code>W.popitem()</code>	
pop(key): to remove an element at key and get its value	<code>print(W.pop(2,-3))</code> <code>print(W.pop(20,-3))</code>	'tues' and W is changed to <code>W={1:'mon', 3:'wed',4:'thurs',5:'fri'}</code> -3

Output??

- `d1={i:i**3 for i in range(1,6)}; print(d1)`
- `{1: 1, 2: 8, 3: 27, 4: 64, 5: 125}`

Output??

- `l=list(); n=6`
- `for i in range(1,n+1):`

`l.append([i,pow(i,3)])`
- `cubesdict=dict(l); print(cubesdict)`
- `{1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216}`

Output??

- `stud={'A':[20,40],'B':[50,60],'C':[90,60]}`
- `for key1,val1 in stud.items():`

```
    print(key1,sum(val1),max(val1))
```

A 60 40

B 110 60

C 150 90

Output?

- d1={'Delhi':10,'Mumbai':8,'Patna':6,'calcut':12}
- for key,val in d1.items():

```
    print(key,'\t','#'*val)
```

#cities population in lakhs

Frequency chart as output:

Delhi #####

Mumbai #####

Patna #####

calcut #####

Difference in tuples/list/dictionary

Tuples	LIST	DICTIONARY
IMMUTABLE	MUTABLE	MUTABLE
Fixed size	Varying size	Varying size
Elements are accessed thru indexes	Elements are accessed thru indexes	Elements are accessed thru keys, no indexes
Ordered	ordered	unordered
Duplicate enteries allowed	Duplicates allowed	Duplicate not allowed

Format()

- for handling complex string formatting more efficiently

Example 1:

- `str = "This is written in {}"`
- `print (str.format("Python")) => This is written in Python`

Example 2:

```
print("The temperature today is {0:.2f} degrees outside !" .format(35.567))
```

```
print("The {0} of 100 is {1:b}".format("binary", 100))
```

```
template = '{0:.2f} {1:s} are worth US${2:d}'
```

```
s=template.format(40.5560, 'Good morning', 1)
```

```
print(s)
```

Output:

The temperature today is 35.57 degrees outside !

The binary of 100 is 1100100

40.56 Good morning are worth US\$1