REC

Functions

Functions, Built-In Functions, Commonly Used Modules, Function Definition and Calling the Function, The return Statement and void Function,

```
def ppsession(, , ):
        '''
        '''
ppsession( , ,)
```

You

REC

Functions

- You can define functions to provide the required functionality. Here are simple rules to define a function in Python.
  - Function blocks begin with the keyword *def* followed by the function name and parentheses ( ).
  - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
  - The first statement of a function can be an optional statement - the documentation string of the function or docstring.
  - The code block within every function starts with a colon (:) and is indented.
  - The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

REC

Functions

- Functions are used when you have a block of statements that needs to be executed multiple times within the program. Rather than writing the block of statements repeatedly to perform the action, you can use a function to perform that action.
- This block of statements are grouped together and is given a name which can be used to invoke it from other parts of the program.
- You write a function once but can execute it any number of times you like.
- Functions also reduce the size of the program by eliminating rudimentary code. Functions can be either Built-in Functions or User-defined functions.

You

A Few Built-in Functions in Python

Function Name	Syntax	Explanation
abs()	*abs(x)* where x is an integer or floating-point number.	The *abs()* function returns the absolute value of a number.
min()	*min(arg_1, arg_2, arg_3,..., arg_n)* where arg_1, arg_2, arg_3 are the arguments.	The *min()* function returns the smallest of two or more arguments.
max()	*max(arg_1, arg_2, arg_3,...,arg_n)* where arg_1, arg_2, arg_3 are the arguments.	The *max()* function returns the largest of two or more arguments.
divmod()	*divmod(a, b)* where a and b are numbers representing numerator and denominator.	The *divmod()* function takes two numbers as arguments and return a pair of numbers consisting of their quotient and remainder. For example, if a and b are integer values, then the result is the same as (a // b, a % b). If either a or b is a floating-point number, then the result is (q, a % b), where q is the whole number of the quotient.
pow()	*pow(x, y)* where x and y are numbers.	The *pow(x, y)* function returns x to the power y which is equivalent to using the power operator: x**y.
len()	*len(s)* where s may be a string, byte, list, tuple, range, dictionary or a set.	The *len()* function returns the length or the number of items in an object.

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

https://docs.python.org/3/library/functions.html

https://www.w3schools.com/python/python_ref_functions.asp

Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

REC

A Few Built-in Functions in Python

Function Name	Syntax	Explanation
abs()	*abs(x)* where x is an integer or floating-point number.	The *abs()* function returns the absolute value of a number.
min()	*min(arg_1, arg_2, arg_3,…, arg_n)* where arg_1, arg_2, arg_3 are the arguments.	The *min()* function returns the smallest of two or more arguments.
max()	*max(arg_1, arg_2, arg_3,…,arg_n)* where arg_1, arg_2, arg_3 are the arguments.	The *max()* function returns the largest of two or more arguments.
divmod()	*divmod(a, b)* where a and b are numbers representing numerator and denominator.	The *divmod()* function takes two numbers as arguments and return a pair of numbers consisting of their quotient and remainder. For example, if a and b are integer values, then the result is the same as (a // b, a % b). If either a or b is a floating-point number, then the result is (q, a % b), where q is the whole number of the quotient.
pow()	*pow(x, y)* where x and y are numbers.	The *pow(x, y)* function returns x to the power y which is equivalent to using the power operator: x**y.
len()	*len(s)* where s may be a string, byte, list, tuple, range, dictionary or a set.	The *len()* function returns the length or the number of items in an object.

You

REC

Built-In Functions

```
>>> abs(-3)
3
>>> min(1, 2, 3, 4, 5)
1
>>> max(4, 5, 6, 7, 8)
8
>>> divmod(5, 2)
(2, 1)
>>> divmod(8.5, 3)
(2.0, 2.5)
>>> pow(3, 2)
9
>>> len("Japan")
5
```

The divmod() function takes two numbers as arguments and return a pair of numbers consisting of their quotient and remainder.

You

Commonly Used Modules

- Modules in Python are reusable libraries of code having *.py* extension, which implements a group of methods and statements. Python comes with many built-in modules as part of the standard library.
- To use a module in your program, import the module using *import* statement. All the *import* statements are placed at the beginning of the program.
- The syntax for import statement is, Keyword *import module_name*

For example, you can import the *math* module as

- 1. >>>import math

REC

dir()

- The built-in function *dir()* returns a sorted list of comma separated strings containing the names of functions, classes and variables as defined in the module.

1. >>> dir(math)

['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp','expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

>>> print(math.sqrt(4))

You

REC

help()

- Another built-in function you will find useful is the *help()* function which invokes the built-in help system.

```
>>> help(math.gcd)
Help on built-in function gcd in module math:
gcd(...)
gcd(x, y) -> int
greatest common divisor of x and y
```

You

Random module

Another useful module in the Python standard library is the *random* m
which generates random numbers.

- The syntax for *random.randint()* function is *random.randint(start, sto*
  which generates a integer number between start and stop argument

numbers (including both)

```
>>> import random
>>> print(random.random())
0.2551941897892144
>>> print(random.randint(5,10))
9
```

You

REC

Import()

The syntax for using a function defined in a module is,

module_name.function_name()

The module name and function name are separated by a dot.

Here we list some of the functions supported by *math* module.

```
>>> import math
>>> print(math.ceil(5.4))
6
>>> print(math.sqrt(4))
2.0
>>> print(math.pi)
3.141592653589793
```

You

REC

pip

Below code shows a simple usage of arrow module.

```
C:\> pip install arrow
>>> import arrow
>>> a = arrow.utcnow()
>>> a.now()
<Arrow [2017-12-23T20:45:14.490380+05:30]>
```

You

Function Definition and Calling the Function

- You can create your own functions and use them as and where it is needed. User-defined function ns are reusable code blocks created by users to perform some specific task in the program.
- The syntax for function definition is,

Keyword

User defined

Parameters

def function_name(parameter_1, parameter_2, …, parameter_n):

Indentation

statement(s)

Colon should be present at the end