DIP: Python

Presenter: Dr. Ha Viet Uyen Synh.

INTRODUCTION

Agenda

Step 1
- Simplicity/ Orthogonality

Step 2
- Syntax Design

Step 3
- Data types & structures

Step 4
- Control Structures

Create your First Python Program

Step 1) ***Open PyCharm Editor***. You can see the introductory screen for PyCharm. To create a new project, click on “Create New Project”.

Step 2) You will need to ***select a location***.

You can select the location where you want the project to be created. If you don’t want to change location than keep it as it is but at least change the name from “untitled” to something more meaningful, like “FirstProject”. PyCharm should have found the Python interpreter you installed earlier. Next Click the “Create” Button.

Step 3) Now Go up to the “File” menu and ***select “New”***. Next, select “Python File”.

Step 4) A new pop up will appear. Now ***type the name of the file*** you want (Here we give “HelloWorld”) and hit “OK”.

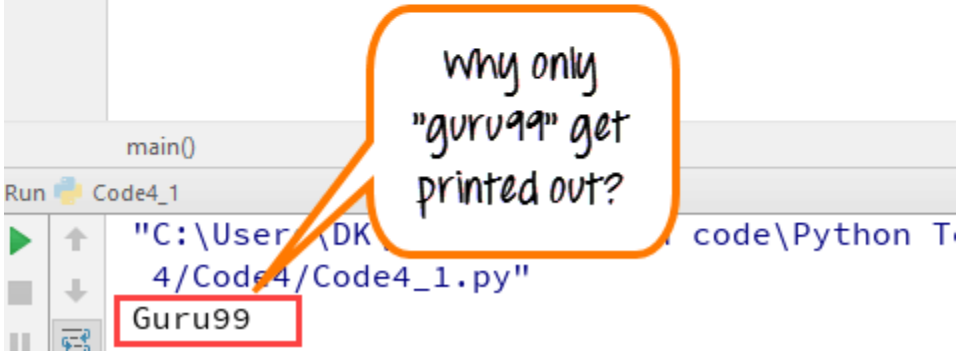
Step 5) Now ***type a simple program*** - print (‘Hello World!’).

Step 6) Now Go up to the “Run” menu and ***select “Run”*** to run your program.

Step 7) You can see the output of your program at the bottom of the screen.

Learn Python Main Function

```
1  def main():
2      print("Hello World!")
3  print("Guru99")
```



It is because we did not declare the call function "if__name__== "__main__".

When Python interpreter reads a source file, it will execute all the code found in it.

When Python runs the "source file" as the main program, it sets the special variable (__name__) to have a value ("__main__").

When you execute the main function, it will then read the "if" statement and checks whether __name__ does equal to __main__.

In Python "if__name__== "__main__" allows you to run the Python files either as reusable modules or standalone programs.

Like C, Python uses == for comparison while = for assignment. Python interpreter uses the main function in two ways

```python
def main():
    print("Hello World!")


if __name__ == "__main__":
    main()

print("Guru99")
```

```
Run  Code4_2
"C:\Users\DK\Desktop\Python code\
 4/Code4/Code4_2"
Hello World!
Guru99
```

Once you define the main function, it will call main function and print "hello world" as well

Syntax

Indentations

Python uses indentation to indicate a block of code.

Example:

```
#Correct
if 5 > 2:
  print("Five is greater than two!")

#Wrong
if 5 > 2:
print("Five is greater than two!")
```

Syntax

Comments

```
#This is a comment.
print("Hello, World!")
```

```
"""This is a
multiline docstring."""
print("Hello, World!")
```

VARIABLES

Variables

Creating Variables

Variables do not need to be declared with any particular type and can even change type after they have been set.

Variable Names

Remember that variables are case-sensitive

A variable name must start with *a letter* or the *underscore character*

A variable name cannot start with a number

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Output Variables

print statement is often used to output variables.

Re-declare a variable

```
1   # Declare a variable and initialize it
2   f = 0
3   print(f)
4   # re-declaring the variable works
5   f = 'guru99'
6   print(f)
7
8
9
10
11
12
```

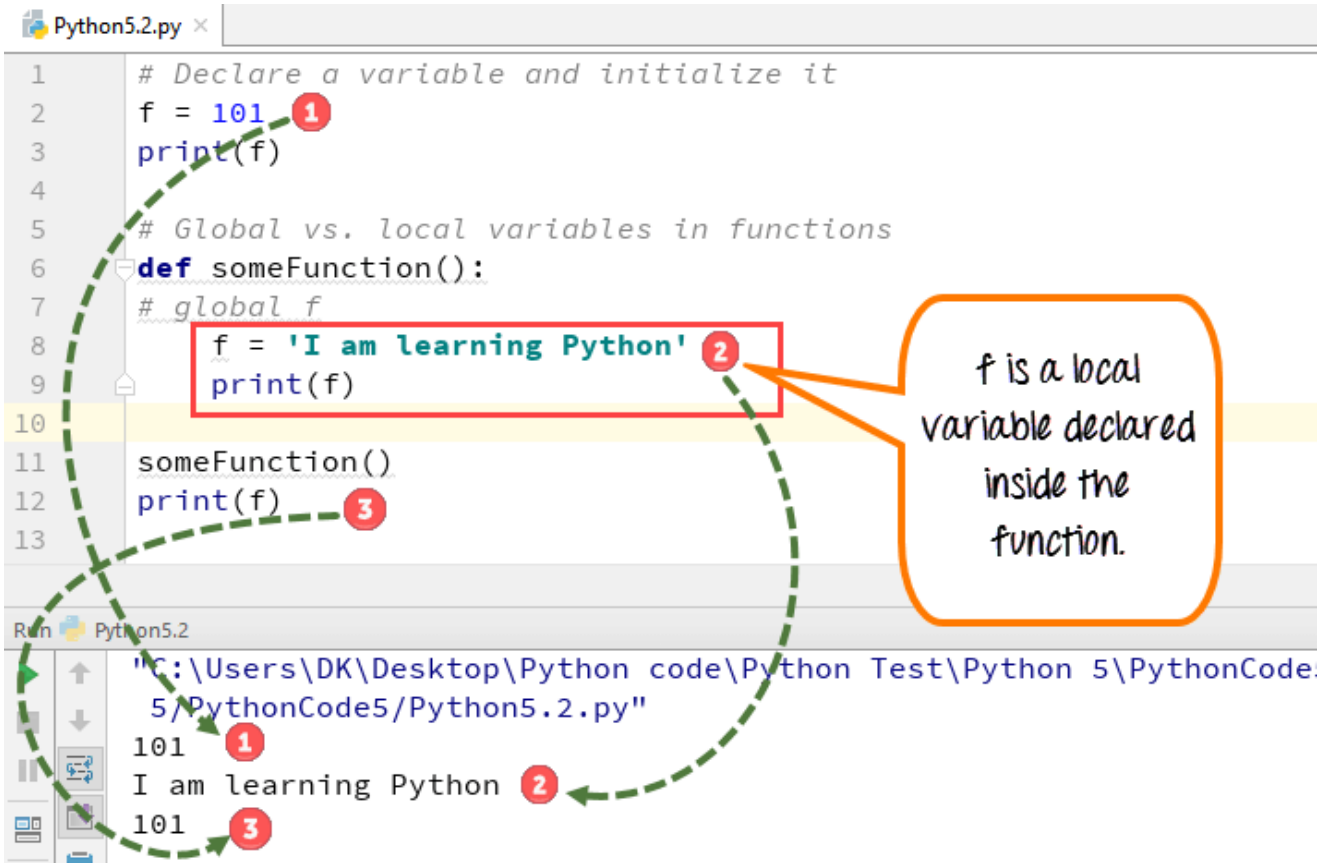
Run Python5.1

"C:\Users\DK e thon Test\Py
5/PythonC de5/

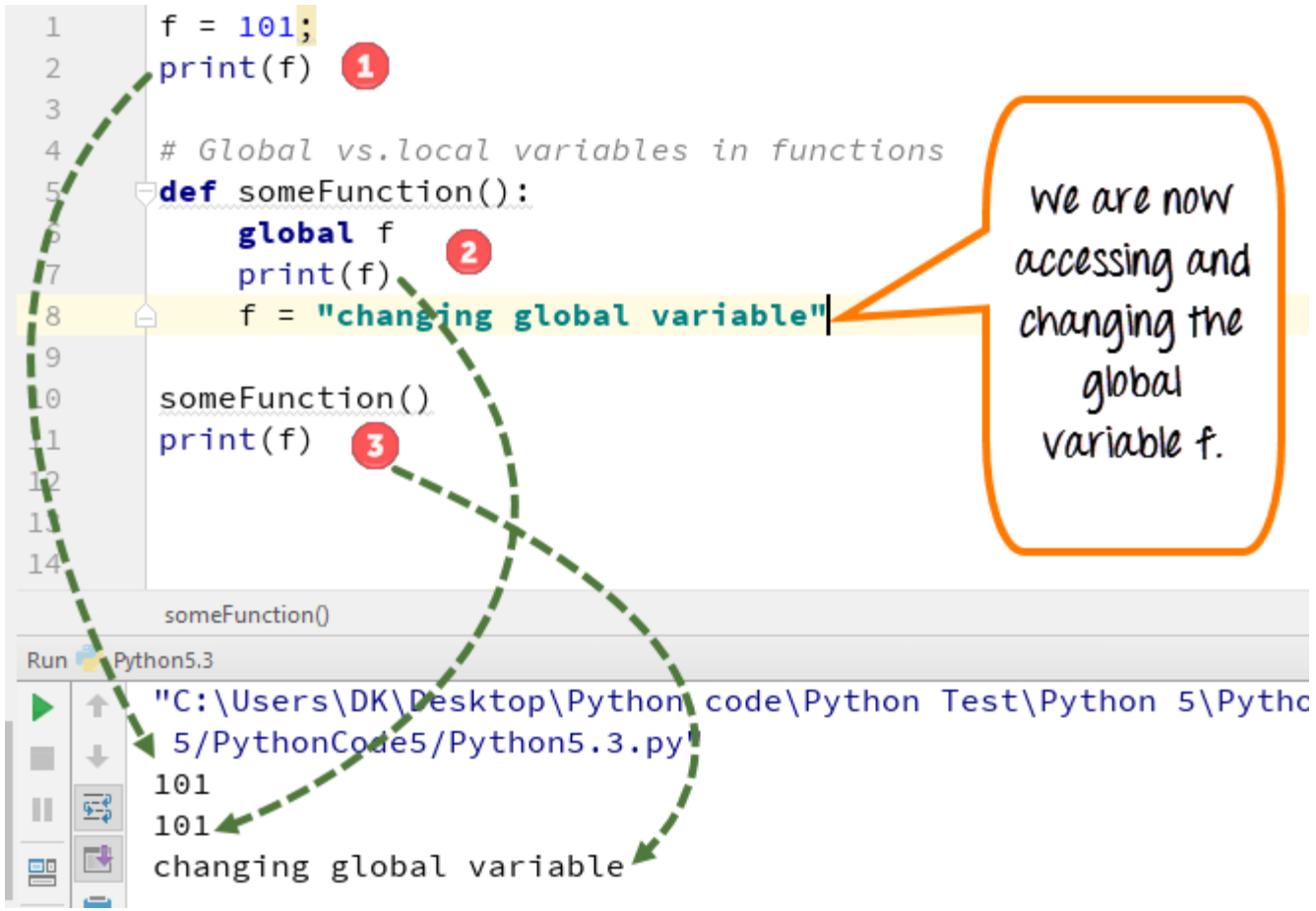
0
guru99

you can re-declare the variables, even-after if it is declared once. it works fine

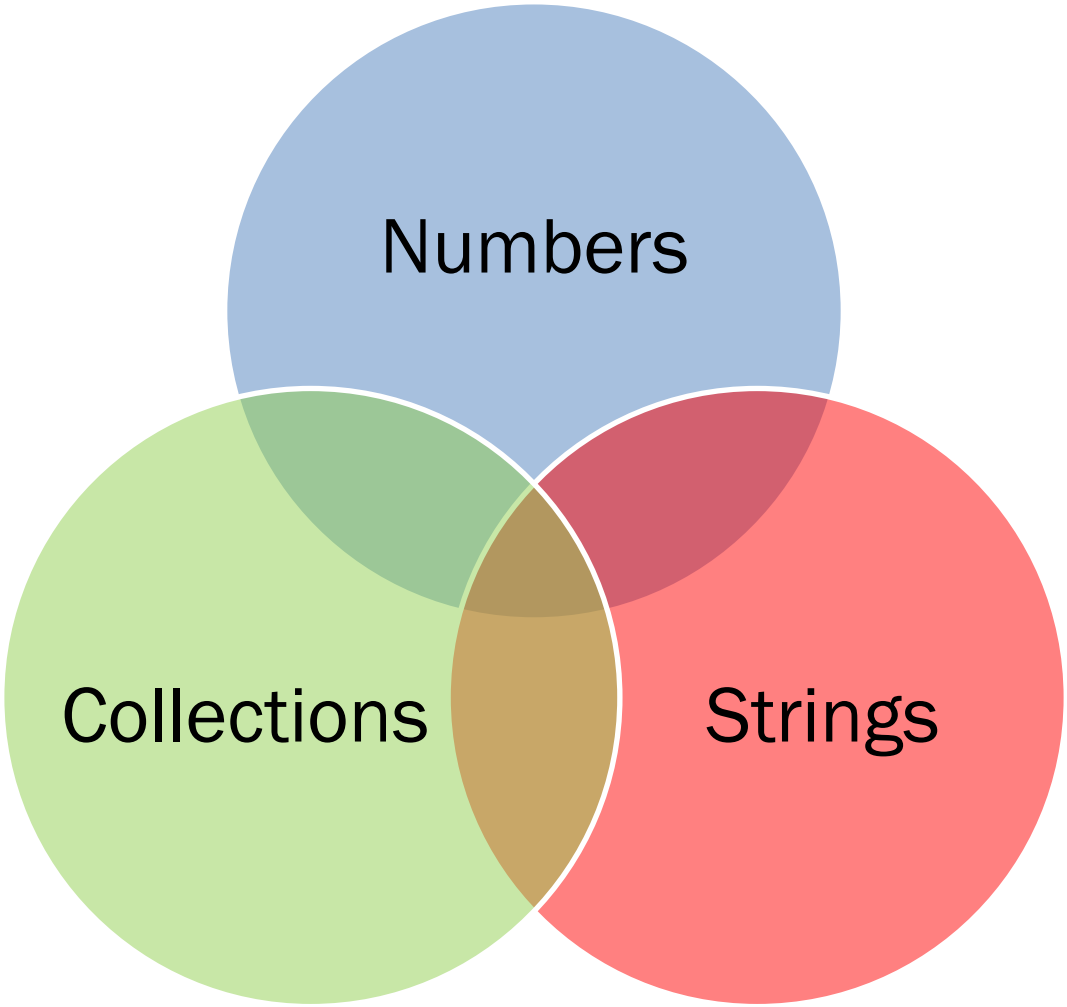
Local & Global Variables



Local & Global Variables



Data types in Python



1. Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them

To verify the type of any object in Python, use the type() function

Ex:

```
x = 1.10
y = 1.0
z = -35.59
print(type(x))
print(type(y))
print(type(z))
```

2. String

String literals in python are surrounded by either single quotation marks, or double quotation marks. ('hello' is the same as "hello“)

Strings can be output to screen using the print function.

Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Ex:

- Get the character at position 1:

```
a = "hello"
print(a[1]) #e
```

- Get the characters from position 2 to position 5:

```
b = "world"
print(b[2:5]) #rld
```

String

- The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!“
```

- The len() method returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

- The lower() method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

- The upper() method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

String

- The replace() method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

- The split() method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

- Input a string from keyboards

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

3. Collections

There are four collection data types in the Python programming language:

- *List* is a collection which is ordered and changeable. Allows duplicate members.
- *Tuple* is a collection which is ordered and unchangeable. Allows duplicate members.
- *Set* is a collection which is unordered and unindexed. No duplicate members.
- *Dictionary* is a collection which is unordered, changeable and indexed. No duplicate members.

3.1 List

A list is a collection which is ordered and changeable.

Create a List:

```
thislist = ["apple", "banana", "cherry"]
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
```

Change an item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
```

Using the append() method to append an item:

```
thislist = list(("apple", "banana", "cherry"))
thislist.append("damson")
```

Using the remove() method to remove an item:

```
thislist = list(("apple", "banana", "cherry"))
thislist.remove("banana")
```

The len() method returns the number of items in a list:

```
thislist = list(("apple", "banana", "cherry"))
print(len(thislist))
```

List methods

append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list, to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

3.2 Tuples

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
```

Return the item in position 1:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

The len() method returns the number of items in a tuple:

```
thistuple = tuple(("apple", "banana", "cherry"))
print(len(thistuple))
```

Note: You cannot remove items in a tuple.

3.3 Sets

A set is a collection which is unordered and unindexed. In Python, sets are written with curly brackets.

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
```

Using the add() method to add an item:

```
thisset = set(("apple", "banana", "cherry"))
thisset.add("damson")
```

Using the remove() method to remove an item:

```
thisset = set(("apple", "banana", "cherry"))
thisset.remove("banana")
```

Using the len() method to return the number of items:

```
thisset = set(("apple", "banana", "cherry"))
```

3.4 Dictionaries

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Create and print a dictionary:

```
thisdict =    {  "apple": "green",
  "banana": "yellow",
  "cherry": "red"
}
thisdict =    dict(apple="green", banana="yellow", cherry="red")
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
```

Change the apple color to "red":

```
thisdict =    {
  "apple": "green",
  "banana": "yellow",
  "cherry": "red"
}
thisdict["apple"] = "red"
```

Dictionaries

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
thisdict =  dict(apple="green", banana="yellow", cherry="red")
thisdict["damson"] = "purple“
```

Removing a dictionary item must be done using the del() function:

```
thisdict =  dict(apple="green", banana="yellow", cherry="red")
del(thisdict["banana"])
```

The len() function returns the size of the dictionary:

```
thisdict =  dict(apple="green", banana="yellow", cherry="red")
print(len(thisdict))
```

Arrays

Arrays are used to store multiple values in one single variable.
Python does not have built-in support for Arrays, but Python Lists can be used instead.

```
cars = ["Ford", "Volvo", "BMW"]
```

Get the value of the first array item:

```
x = cars[0]
```

Modify the value of the first array item:

```
cars[0] = "Toyota“
```

```
x = len(cars)
```

Add one more element to the cars array:

```
cars.append("Honda")
```

Delete the second element of the cars array:

```
cars.pop(1)
cars.remove("Volvo")
```

Array Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Operators in Python

Arithmetic operators

Bitwise operators

Assignment operators

Membership operators

Comparison operators

Identity operators

Logical operators

Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity Operators

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns false if both variables are the same object	x is not y

Membership Operators

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns False if a sequence with the specified value is present in the object	x not in y

Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

CONTROL STRUCTURES

Conditions

If statement:

```
a = 33
b = 200
if b > a: print("b is greater than a")
```

Elif statement:

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else statement:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

While Loops

While statement:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Break statement:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Continue statement:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

For Loops

For statement

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Break statement

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Continue statement

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

range() function

```
for x in range(6):
  print(x)
for x in range(2, 6):
  print(x)
for x in range(2, 30, 3):
  print(x)
```

MODULUS

Functions

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
A function can return data as a result.

Creating a Function

```
def my_function():
  print("Hello from a function")
my_function()
```

Parameters

```
def my_function(fname):
  print(fname + " Refsnes")
my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Functions

Default Parameter Value

```
def my_function(country = "Norway"):
  print("I am from " + country)
my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Return Values

```
def my_function(x):
  return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Lambda Functions

The keyword lambda is used to create what is known as anonymous functions.

```
myfunc = lambda i: i*2
print(myfunc(2))

myfunc = lambda x,y: x*y
print(myfunc(3,6))

def myfunc(n):
  return lambda i: i*n
doubler = myfunc(2)
tripler = myfunc(3)
val = 11
print("Doubled: " + str(doubler(val)) + ". Tripled: " + str(tripler(val)))
```

Recursion

```
def tri_recursion(k):
  if(k>0):
    result = k+tri_recursion(k-1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

Classes

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

Create a class named Person, use the *__init__()* function to assign values for name and age. The __init__() function is called automatically every time the class is being used to create a new object.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

Classes

Insert a function that prints a greeting, and execute it on the p1 object. The self parameter is a reference to the class itself, and is used to access variables that belongs to the class.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def myfunc(self):
    print("Hello my name is " + self.name)
p1 = Person("John", 36)
p1.myfunc()
```

Classes

The self parameter is a reference to the class itself, and is used to access variables that belongs to the class.

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age
  def myfunc(abc):
    print("Hello my name is " + abc.name)
p1 = Person("John", 36)
p1.myfunc()
```

Set the age of p1 to 40:

```
p1.age = 40
```

Delete the age property from the p1 object:

```
del p1.age
```

Delete the p1 object:

```
del p1
```

Questions? More Information?

✉ hvusynh@hcmiu.edu.vn