Functions

Lecture #4 Notes – Python - Name ________ Class __________

1) Functions are stored and reused steps
    a. TRY THIS PROGRAM

```
def hello():
   print('Hello')
   print('fun')
hello()
print ('zip')
hello()
```

2) In the context of programming, a **function** is a <u>named sequence of statements</u> that <u>performs a computation</u>. When you define a function, you specify the name and the sequence of statements. Later, you can “<u>call</u>” the function by name. We have already seen one example of a **function call**:

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

The name of the function is type. The expression in parentheses is called the <u>argument</u> of the function. The argument is a <u>value or variable</u> that we are passing into the function as <u>input</u> to the function. The result, for the type function, is the type of the argument.

It is common to say that a function “<u>takes</u>” an argument and “<u>returns</u>” a result. The result is called the <u>return value</u>.

<u>PROGRAM</u>

```
print(max('Hello world'))

print(min('TYPEYOURNAMEHERENOSPSACES'))
```

3) You should treat the names of built-in functions as <u>reserved words</u> (i.e., avoid using “max” as a variable name).

```
print(len('house'))
```

4) Random numbers

```
import random
for i in range(10):
   x = random.random()
   print (x)
```

5) TRY THIS IN THE SHELL

```
import math
print (math)
```

6)
The rules for function names are the same as for <u>variable</u> names: letters, numbers and some punctuation marks are legal, but the first character can’t be <u>a number</u>. You can’t use a <u>keyword</u> as the name of a function, and you should avoid having a variable and a function with the same name.

7) The first line of the function definition is called the **<u>header</u>**; the rest is called the **<u>body</u>**.

TRY THIS WITH YOUR OWN LYRICS – notice below a function inside of a function

```
def print_lyrics():

   print ("I'm a lumberjack, and I'm okay.")

   print ("I sleep all night and I work all day.")

print_lyrics()

def repeat_lyrics():

   print_lyrics()

   print_lyrics()
```

```
repeat_lyrics()
```

8) When you read a program, you don’t always want to read from top to bottom. Sometimes it makes more sense if you follow the **flow of execution**.

9) BELOW THE PARAMETER IS x AND THE VALUE OF THE PARAMTER IS doug OR WHATEVER YOU PUT INTO IT – TRY IT

```
def print_twice(x):
   print (x)
   print (x)
print_twice('doug')
```

10)

Some of the functions we are using, such as the math functions, yield results; for lack of a better name, I call them **fruitful functions**. Other functions, like `print_twice`, **perform an action** but don’t return a **value**. They are called **void functions**.

TRY IN THE SHELL:

```
>>> result = print_twice('Bing')
```

THEN

```
>>> print(result)
None
```

TRY IN THE SHELL:

```
print (type(None))
```

11) To return a result from a function, we use the **<u>return</u>** statement in our function. For example, we could make a very simple function called addtwo that adds two numbers together and returns a result.

PROGRAM

```
def addtwo(a, b):
   added = a + b
   return added

x = addtwo(3, 5)
print x
```

12) **Why functions?**

It may not be clear why it is worth the trouble to divide a program into functions. There are several reasons:

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

Work on Ex 4.14 in online textbook

Score : ____ / 10 Answers
____ / 10 Participation / Attitude