Python State Test Study Guide

Programming I - Python

"Everything you should know as a beginning python programmer"

Mr. Alvey

Variables - naming rules and conventions

Variables: names (or **identifiers**) we assign to data so we can access it. Example: **number** = 53, **name** = "Monty Python"

*the assignment operator (=) is used to assign data values to variables. remember it is only one equal sign.

Rules for naming variable:

1. No symbols, only letters and numbers
   (The underscore _ is the only exception)
2. Can't start with a number
3. No spaces
4. No python reserved words (i.e. if, else, def, etc.)

Naming Conventions:

5. descriptive identifier names for readability
6. start variables with lowercase letters

Data types you should know

Integers (int)
Floating point (float)
String (str)
Boolean (bool)
Long Integers (long)
Lists [ ]

Extra links for study:
http://diveintopython3.org/native-datatypes.html
http://www.tutorialspoint.com/python/python_variable_types.htm

Data Types - Numbers

Two main types of numbers: **Integer** and **Floating point**

- Integers (**int**)
  - Integers are plain numbers with no decimals (0, 1, 34, 4893, 23, 12)
  - assigning integers to variables:

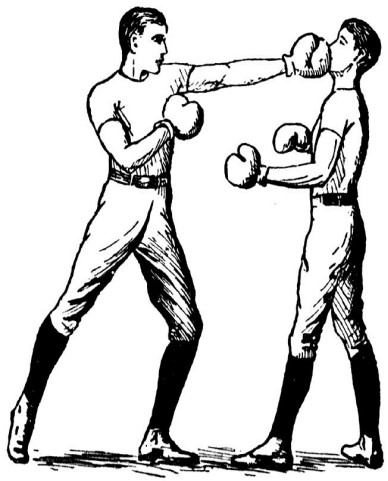
```
num = 32
num1, num2 = 12, 14
```

- Floating point (**float**)
  - Floating point numbers have a decimal point (0.5, 0.67, 5.23, 6.0, 423.2342342, 78.)
  - assigning integers to variables:

```
num = 5.2
mygrade, average = 89.2, 67.9
```

Data Types - Numbers

Floating point VS. Integer



Be aware of Floating Point division!

- An integer divided by an integer is an integer !!!!!!!
  5/3 is 1, 22/5 is 4, 2/3 is 0, 1/2 is 0
- A float divided by an integer (or visa versa) will be a float
  5.0/3 is 1.6, 22/5.0 is 4.4, 1./2 is 0.5

```
>>> 5/3
1
>>> 5.0/3
1.6666666666666667
>>> 5./3
1.6666666666666667
```

```
>>> 1/2
0
>>> 1/2.
0.5
>>> 1.0/2.0
0.5
```

Data Types - Strings

String: A list of characters strung together to make words; strings are characters enclosed in quotes.

Examples:

- "Hello World"
- "x"
- "This is a long string of characters including spaces"
- 'Hello World'
- 'Single quotes work in python too'

Assigning a string to a variable:

```
>>> name = "James Durbin"
>>> street = 'Sesame Street'
>>> color = "green"
```

Data Types - String indexing and slicing

Indexing: Selecting a character in a string by index number.

```
>>> letters = "abcdefghijklmnop"
>>> letters[0]
'a'
>>> letters[5]
'f'
```

Slicing: Selecting smaller strings from a longer string

```
>>> sentence = "disfunctional"
>>> sentence[1:3]
'is'
>>> sentence[3:6]
'fun'
```

Data Types - Fun with Slicing

```
>>> secret = "sdrawkcab"
>>> secret[::-1]
'backwards'
>>> hidden = "smcehestl padtb ktweunt"
>>> hidden[1::2]
'meet at ten'
```

Three sections for slicing **string**
[1:2:3]
1. starting index
2. ending index
3. increment (count by this many)

Data Types - Lists

List: a group of data types that can be indexed
Example: students = ["Jacob", "Logan", "Brad", "Kevin"]

A list allows you to group similar data using a single variable name. Then you can access the data by indexing and slicing.

```
>>> states = ['UT','NV','ID','WY','TX','CO','CA']
>>> states[0]
'UT'
>>> states[3:6]
['WY', 'TX', 'CO']
>>> states[::2]
['UT', 'ID', 'TX', 'CA']
```

*Lists are very easy to use with for loops

Data Types - List functions

Lists have two functions you should be aware of:

1. **append**, adds data to the list at the end
2. **insert**, adds data to the list at a specific index

```
>>> genres = []
>>> genres.append('Jazz')
>>> genres.append('Rap')
>>> print genres
['Jazz', 'Rap']
>>> genres.insert(1,'Pop')
>>> print genres
['Jazz', 'Pop', 'Rap']
>>> genres.insert(1,'Celtic')
>>> print genres
['Jazz', 'Celtic', 'Pop', 'Rap']
```

Operators - Arithmetic

Operators in python are symbols or words that do things. The basic arithmetic operators are listed.

+ Addition operator
- Subtraction operator
/ Division operator
* Multiplication operator
** Exponents operator
% Modulus operator (remainder)

```
>>> num1, num2 = 5, 7
>>> 3 + 73 - num1 / 2 * num2
62
>>> 28 % 3
1
>>> 5 ** 3
125
>>> (12**2 + 5**2)**(1/2.0)
13.0
```

Be careful when dividing integers (dividing integers loses decimal values)

```
>>> careful = [1/2, 1/2.0, 3/5, 3./5]
>>> print careful
[0, 0.5, 0, 0.59999999999999998]
```

Operators - Assignments

The assignment operator is the equal sign =

But there are short cuts to do arithmetic and make assignments, here the arithmetic assignment shortcuts:

+= add the value to the variable
-= subtracts the value from the variable
*= multiplies the value by the variable
/= divides the variable by the value

```
>>> number = 1
>>> number += 5
>>> number
6
>>> number -= 2
>>> number
4
>>> number *= 2
>>> number
8
>>> number /= 4
>>> number
2
```

Operators - Comparison

Comparison operators ask True or False questions (Boolean)

== "is equal to?"
> "is greater than?"
>= "is greater than or equal to?"
< "is less than?"
<= "is less than or equal to?"
!= "is not equal to?"

```
>>> num1, num2 = 5, 9
>>> num1 == num2
False
>>> num1 >= num2
False
>>> num1 <= num2
True
```

Operators - Logic

Logic Operators: These operators combine boolean expressions logically using and and or and not

Examples:

```
>>> num1, num2 = 5, 9
>>> num1 == num2 or num1 < num2
True
```

Operators - Logic Chart

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

not True is False
not False is True

Operators - Misc.

Comma , The comma operator is used to separate expressions. When used with the print statement is continues the line.

Examples: print "hello",
num1, num2 = 5, 6
dosomething(name, address, phone)

Concatenation + The concatenation operator is used to combine strings

Example: name = "John" + " " + "Wayne"

Dot . The dot operator allows objects to access data members and functions

Examples: turtle.left() random.randint(10)

Errors - Syntax

Syntax: The program can't run (or can't be interpreted) because the code break the rules

Example: (no colon for the if control structure)

```
>>> a, b = 1, 2
>>> if a > b
  File "<stdin>", line 1
    if a > b
           ^
SyntaxError: invalid syntax
```

Errors - Run-time

Run-time Error: The program runs, but it crashes while it runs.

Examples:

- Dividing by zero
- Index of list too big (out of range)
- missing file
- etc

```
>>> x = [23, 64, 67, 34, 23]
>>> x[7]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

```
>>> for num in [1, 2, 4, 0]:
...     print 10/num
...
10
5
2
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Errors - Logic

Logic Error: The program runs, but it doesn't do what you want it to do.

Example:

- A bank program puts money in the wrong account.
- Paint program translates the wrong color
- In a game the pressing buttons do the wrong things

Error - Linking

Problems compiling the files

Errors - Debugging

Debugging is the process of find errors and fixing them-getting rid of the bugs!

Control Structures

A Control Structure in python is anything that controls the flow of the program. Some code is skipped (if), some code is repeated (loops), and some code is labeled for later use (functions).

A control structure opens with a colon (:). The body of a control structure is always indented.

```
>>> for i in range(10):
...     print i
>>> alvey,jamison = "Awesome","Annoying"
>>> if alvey > jamison:
...     print "Alvey is Awesome!"
...     print "Jamison is Annoying."
```

colon

indent

Control Structures - if / elif / else

The **if** control structure will test a condition (true or false) to decide if a block of code should be run or skipped.

```
>>> score = 67
>>> if score > 90:
...     print "A"
... elif score > 80:
...     print "B"
... elif score > 70:
...     print "C"
... elif score > 60:
...     print "D"
... else:
...     print "F"
...
D
```

Control Structures - While Loops

While loops will test a condition and repeat a block of code until the condition becomes False.

```
>>> count = 10
>>> while count > 0:
...     print count,
...     count -= 1
...
10 9 8 7 6 5 4 3 2 1
```

Control Structures - For Loops

For loops will read a list and repeat a block of code for every item in the list.

```
>>> birds = ['chicken','turkey','peacock','eagle']
>>> for bird in birds:
...     print bird,
...
chicken turkey peacock eagle
```

```
>>> for number in range(10):
...     print number,
...
0 1 2 3 4 5 6 7 8 9
```

Control Structures - Functions

Functions define small blocks of code that can be called at any point in a program.

(function definition)

```
>>> def novowels(word):
...     new_word = ""
...     for letter in word:
...         if letter not in "aeiouy":
...             new_word += letter
...     return new_word
```

(funciton calls)

```
>>> novowels("mississippi")
'mssspp'
>>> novowels("beautifully")
'btfll'
```

Accumulator Variable

When using loops we often need a variable to keep track of an accumulating (increasing) number, like a total. This variable should be defined outside the loop and changed in the loop. Here is an example:

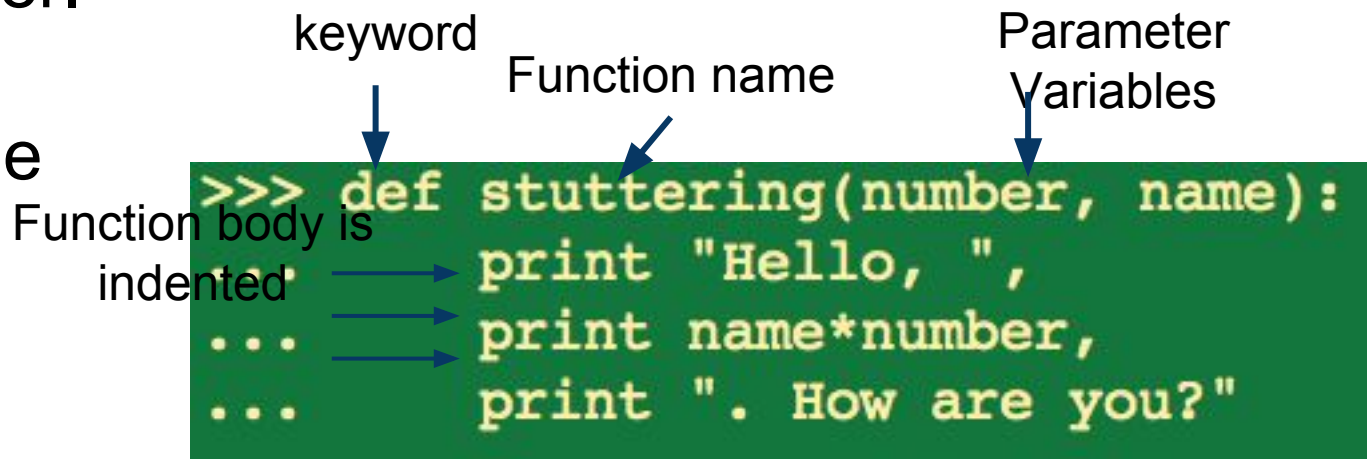
```
>>> #accumulator variable
... total = 0
>>> while total < 10:
...     print "No Games"
...     total += 1
```

When total accumulates to 10 the loop will end and the total variable now has the value 10.

Functions - Defining

Define a function using the key word **def**. A function defines a block of code that can be re-used in your program. Designing your program from the **top down** using functions is a common best practice in programming. Each level you generalize what you want to happen with a function name, then define the function details later.

Here is an example of a how to define a function:



Functions - Calling

After a function is defined, it won't do anything until the function is called by its name. When a function is called by its name and all the arguments are given to the parameter variables, the code in the function's body is run.

Here are a few examples of calling the stuttering function:

Call the function by its name

Give argument values to the parameters

```
>>> stuttering(3, "Betty")
Hello,  BettyBettyBetty . How are you?
```

```
>>> stuttering(5, "Bob")
Hello,  BobBobBobBobBob . How are you?
```

The function outputs the results of its code body

Functions - Return

When a function is called the interpreter "moves" to the function's code, and when the function's code is finished it "returns" to where it was called. The keyword **return** can be used to end a function and "return" it. If a value is given the function can return a value when it is called.

Example:

```
>>> def countAs(word):
...     total = 0
...     for letter in word:
...         if letter == 'a':
...             total += 1
...     return total
```

returns the total number of A's in the word.

The return value can be stored in a variable.

```
>>> countAs("Banana")
3
>>> answer = countAs("Bananarama")
>>> answer
5
```

The `range()` function

The **range()** function is used for creating lists of numbers in a specific range. **Use the range() function with for loops to count.**

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> for num in range(10):
...     print num,
...
0 1 2 3 4 5 6 7 8 9
```

The 3 arguments for the range(**arg1**,**arg2**,**arg3**) function are

- arg1 is the starting number
- arg2 is the ending number
- arg3 is the step

```
>>> range(5,15)
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> range(5,50,5)
[5, 10, 15, 20, 25, 30, 35, 40, 45]
>>> range(10,0,-1)
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Misc. things you should know

- The **import** statement, and sometime **from** (know the difference between them)
  import turtle
  from turtle import *
- The open() function for opening files. "r" for read "w" for write. Give it the name of the file. This code reads all the lines in a file.
  file = open("names.txt","r") for line in file: print line file.close()

Misc. things you should know

- The **print** statement is for output. Use the comma operator to continue a line
- Use conversion functions to convert variable from one data type to another.
  str() - convert numbers to a string
  int() - convert a string or a float to an integer
  float() - convert an integer to a float

- The **len()** function will give you the length of a string or a list
  `len("jeremy")` outputs 6
  `len(["bobby", "brad", "benny"])` outputs 3

- Sequential search: Search a long list one item at a time using a loop. Search a list in "sequence".

Misc. things you should know

- Converting Decimal numbers to Binary
  12 -> 1100
  127 -> 1111111
- Converting Binary numbers to Decimal
  110110 -> 54
  11101 -> 29
- **Boolean** means True/False or 1/0 or On/Off

String Formatting

String formatting is a shortcut for outputting strings with variable values inserted.

- %s is a place holder for string variables
- %d is a place holder for number variables

Example:

```
name = "Bob"
place = "Brazil"
age = 28
```

Place Holders for variables

Separator

Variables (used in order)

```
"My name is %s, I am from %s, I am %d yrs old" % (name, place, age)
```

output:

```
My name is Bob, I am from Brazil, I am 28 yrs old
```

Object Oriented Programming (OOP)

Object Oriented Programming: Creating data types that represent objects. These objects have properties (data attributes) and functions (or methods). In this way the programmer can think of the code in objects and what they look like and what they can do.

Example: I could create a dog object and the dog might be brown with spots and the dog could walk or run

```
fido = dog("brown", "spots")
fido.run()
fido.walk()
```

OOP - Class definition

The keyword **class** is used to define the blue prints for creating a new object data type. The class definition is used each time a new object is created.

keyword

Object data type name

```
>>> class dog:
...     def __init__(self, color="brown", spots=True):
...         self.color = color
...         self.spots = spots
...     def run(self):
...         self.speed = "Fast"
...         return self.speed
...     def walk(self):
...         self.speed = "Slow"
...         return self.speed
```

properties or data attributes.

functions or methods

OOP - The dot operator

The little dot . is a powerful operator that is used to access the data and functions of an object.

See how the name of the dog object is fido, and the dot allows fido to access his color, his spots, and it lets him run and walk. That is a cool dot!

```
>>> fido = dog("black")
>>> fido.color
'black'
>>> fido.spots
True
>>> fido.run()
'Fast'
>>> fido.walk()
'Slow'
```

OOP - The __init__() function

When an instance of a class is created, the __init__() function is called. The init function's job is to give initial values to the objects properties (data attributes), like giving the color "white" to a new dog named "rex".

```
>>> rex = dog("white")
>>> rex.color
'white'
```

The dog object's color will default to "brown", unless a color is given. With rex the color "white" is given when rex is created.

Default value

```
def __init__(self, color="brown", spots=True):
    self.color = color
    self.spots = spots
```

properties are assigned values when a new object instance is created

OOP - Self

self is a special variable that helps an object refer to it**self**. It is similar to how you and I would use the word "mine". The **self** variable must always be the first variable of a class function definition, and it is used to label properties.

```
>>> class dog:
...     def __init__(self, color="brown", spots=True):
...         self.color = color
...         self.spots = spots
...     def run(self):
...         self.speed = "Fast"
...         return self.speed
...     def walk(self):
...         self.speed = "Slow"
...         return self.speed
```

OOP - Instances

An instance of an object is when you use the class definition to create a new object. Just like you can make multiple houses from one blue print, you can make multiple object instances from one class definition.

Each dog instance has its own properties

```
>>> princess = dog("pink")
>>> rex = dog("white")
>>> fido = dog("blue")
>>> pluto = dog("light brown")
>>> harry = dog("black")
>>> gina = dog("red")
>>> skip = dog("black")
>>> rover = dog("purple")
>>> buddy = dog("grey")
```

Variable Scope

The scope of a variable is how long it exists in memory for your program. A variable only exists in the control structure where it was initially assigned. Or, in other words it only exists at the current indentation level where it was created.

Example: The counter variable is defined in the somefunction() function, but it can't be used outside that function:

```
>>> def somefunction():
...     counter = 7
...     counter += 21
...
>>> counter
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'counter' is not defined
```

Global Scope

When variables are defined on the farthest left edge they are globally defined and can be used by any definition of control structure. The keyword **global** can be used to force a variable to have global scope at any level

Examples:

```
number = 5

if number == 5:
        global bob
```

#Comments

The pound sign # is used for comments in python. Comments are for the programmer to read. They will be skipped by the interpreter

isn't that nice?
why, yes it is.

Source Code

Source Code: Code written in a **high level language** (such as python) that will be translated into machine code (low level languages) and then into pure binary code (1's and 0's) for the computer to read

Software Developer

A software developer is a programmer. Programmers write code to create software programs to make peoples lives easier.

The software developers main job description is to design and write and test the code for software programs, not anything else.

Team work

Working together in a group and sharing the load is a good thing. When code is shared in a team everyone can work on individual functions and get the project finished much faster.

Interpreter vs Compiler

Both interpreters and compilers have the same job--they **convert high level source code into machine code** that can be read by the computer's hardware.

Interpreter is a program that converts the code in real time

Compiler is a program that creates a converted file (takes a minute to compile each time you make a change)

Designing Programs

Plan out your programs and analyze all the program requirements before you jump into coding. A little effort to plan and write flow charts and list the requirement for the program will save lots of time in the long run.