Week 2

Expressions, variables, for loops

Constants

- Fixed values such as numbers, letters, and strings, are called “constants” because their value does not change
- Numeric constants are as you expect
- String constants use single quotes (') or double quotes (?)

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Reserved Words

- You cannot use reserved words as variable names / identifiers

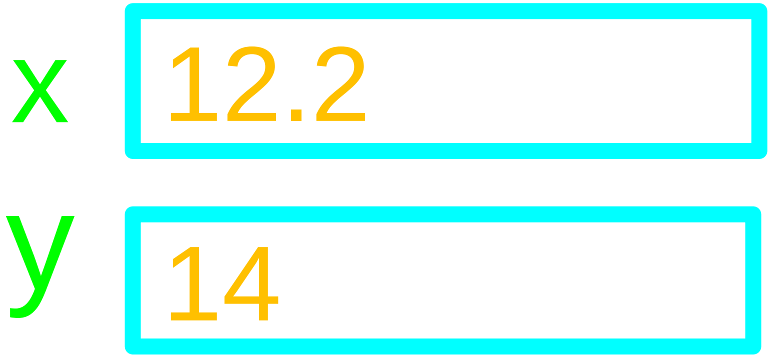
False	class	return	is	finally
None	if	for	lambda	continue
True	def	from	while	nonlocal
And	del	global	not	with
As	elif	try	or	yield
Assert	else	import	pass	
break	except	in	raise	

python™

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name"
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

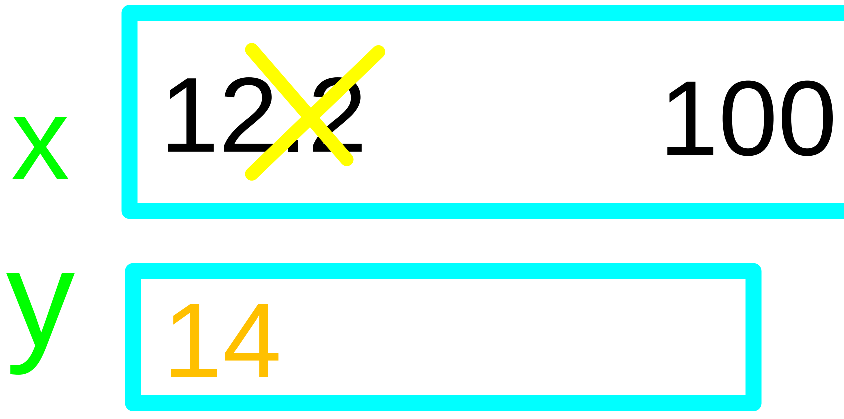
```
x    12.2
y    14
```



Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name"
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
x = 100
```



Expressions

- Arithmetic is very similar to Java
  - Operators: `+ - * / %` (plus `**` for exponentiation)
  - Precedence: `()` before `**` before `* / %` before `+ -`
  - Integers vs. real numbers
  - You may use // for integer division

```
>>> 1 + 1
2
>>> 1 + 3 * 4 - 2
11
>>> 7 // 2
3
>>> 7 / 2
3.5
>>> 7.0 / 2
3.5
```

Operator Precedence Rules

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
```

Parenthesis

Power

Multiplicat on

Additior

Left to Right

```
1 + 2 ** 3 / 4 * 5

1 + 8 / 4 * 5

1 + 2 * 5

1 + 10

11
```

Variables

- Declaring
  - no type is written; same syntax as assignment
- Operators
  - no `++` or `--` operators (must manually adjust by 1)

Java	**Python**

`int x = 2;`
`x++;`
`System.out.println(x);`
	`x = 2`
	`x = x + 1`
	`print(x)`

`x = x * 8;`
`System.out.println(x);`
	`x = x * 8`
	`print(x)`

`double d = 3.2;`
`d = d / 2;`
`System.out.println(d);`
	`d = 3.2`
	`d = d / 2`
	`print(d)`

Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive

```
Good:        spam    eggs    spam23    _speed
Bad:         23spam      #sign   var.12
Different:        spam    Spam    SPAM
```

Mnemonic Variable Names

- We name variables to help us remember what we intend to store in them ("mnemonic" = "memory aid")
- Avoid confuse

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

Types

- Python is looser about types than Java
  - Variables' types do not need to be declared
  - Variables can change types as a program is running

Value	Java type	Python type
`42`	`int`	`int`
`3.14`	`double`	`float`
`"ni!"`	`String`	`str`

What Does “Type” Mean?

- In Python variables, literals, and constants have a “type”
- Python knows the **difference** between an integer number and a string
- For example “+” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```

concatenate = put together

Several Types of Numbers

- Numbers have two main types
- Integers are whole numbers:

-14, -2, 0, 1, 100, 401233

- Floating Point Numbers have

decimal parts: -2.5 , 0.0, 98.6, 14.0

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
>>>
```

Type Conversions

When you put an integer and floating point in an expression, the integer is **implicitly** converted to a float

You can control this with the built-in functions int() and float()

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```

Integer Division

- Integer division produces a floating point result

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

String Conversions

- You can also use int() and float() to convert between strings and integers
- You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>
TypeError: Can't convert 'int'
object to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>
ValueError: invalid literal for
int() with base 10: 'x'
```

Converting User Input

- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function

```
inp = input('Europe floor?')
usf = int(inp) + 1
print('US floor', usf)
```

String Multiplication

- Python strings can be multiplied by an integer.
  - The result is many copies of the string concatenated together.

```
>>> "hello" * 3
"hellohellohello"

>>> print(10 * "yo ”)
yo yo yo yo yo yo yo yo yo yo

>>> print(2 * 3 * "4“)
444444
```

String Concatenation

- Integers and strings cannot be concatenated in Python.
  - Workarounds:
  - `str(`**value**`)` - converts a value into a string
  - `print` **value**`,` **value** - prints value twice, separated by a space

```
>>> x = 4
>>> print("Thou shalt not count to " + x + ".")
TypeError: cannot concatenate 'str' and 'int' objects

>>> print("Thou shalt not count to " + str(x) + ".")
Thou shalt not count to 4.

>>> print(x + 1, "is out of the question.")
5 is out of the question.
```

The `for` Loop

- `for` **name** `in range(`**max**`):`
- **statements**

- Repeats for values 0 (inclusive) to **max** (exclusive)

```
>>> for i in range(5):
...     print(i)
0
1
2
3
4
```

`for` Loop Variations

- `for` **name** `in range(`**min**`,` **max**`):`
-   **statements**

- `for` **name** `in range(`**min**`,` **max**`,` **step**`):`
-   **statements**

- Can specify a minimum other than 0, and a step other than 1

```
>>> for i in range(2, 6):
...     print(i)
2
3
4
5
>>> for i in range(15, 0, -5):
...     print(i)
15
10
5
```

Nested Loops

- Nested loops are often replaced by string * and +

```
....1
...2
..3
.4
5
```

Java

```java
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (5 - line); j++) {
        System.out.print(".");
    }
    System.out.println(line);
}
```

Python

```python
for line in range(1, 6):
    print((5 - line) * "." + str(line))
```

python™

Constants

- Python doesn't really have constants.
  - Instead, declare a variable at the top of your code.
  - All methods will be able to use this "constant" value.

constant.py

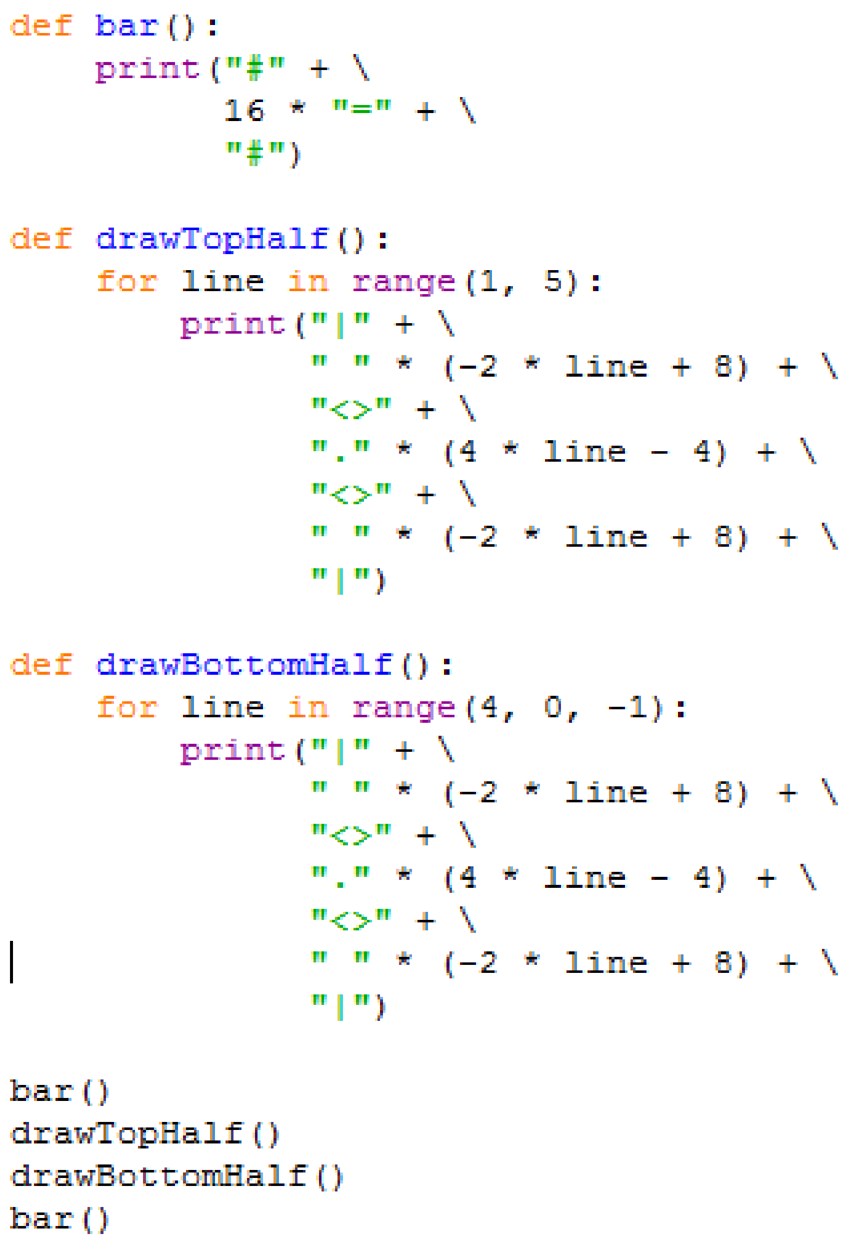
```
MAX_VALUE = 3

def printTop():
    for i in range(MAX_VALUE):
        for j in range(i):
            print(j)
        print()

def printBottom():
    for i in range(MAX_VALUE, 0, -1):
        for j in range(i, 0, -1):
            print(MAX_VALUE)
        print()
```

Exercise

- Rewrite the Mirror lecture program in Python. Its output:

```
#================#
|      <><>      |
|    <>....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |
#================#
```

  - Make the mirror resizable by using a "constant."



```python
def bar():
    print("#" + \
          16 * "=" + \
          "#")

def drawTopHalf():
    for line in range(1, 5):
        print("|" + \
              " " * (-2 * line + 8) + \
              "<>" + \
              "." * (4 * line - 4) + \
              "<>" + \
              " " * (-2 * line + 8) + \
              "|")

def drawBottomHalf():
    for line in range(4, 0, -1):
        print("|" + \
              " " * (-2 * line + 8) + \
              "<>" + \
              "." * (4 * line - 4) + \
              "<>" + \
              " " * (-2 * line + 8) + \
              "|")

bar()
drawTopHalf()
drawBottomHalf()
bar()
```

Exercise Solution

```
SIZE = 4

def bar():
    print("#" + 4 * SIZE * "=" + "#")

def top():
    for line in range(1, SIZE + 1):
        # split a long line by ending it with \
        print("|" + (-2 * line + 2 * SIZE) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 2 * SIZE) * " " + "|")

def bottom():
    for line in range(SIZE, 0, -1):
        print("|" + (-2 * line + 2 * SIZE) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 2 * SIZE) * " " + "|")
# main
bar()
top()
bottom()
bar()
```

Concatenating Ranges

- Ranges can be concatenated with +
  - However, you must use the “list()” command
  - Can be used to loop over a disjoint range of numbers

```
>>> list(range(1, 5)) + list(range(10, 15))
[1, 2, 3, 4, 10, 11, 12, 13, 14]

>>> for i in list(range(4)) + list(range(10, 7, -1)):
...     print(i)
0
1
2
3
10
9
8
```

Exercise Solution 2

```
SIZE = 4

def bar():
    print "#" + 4 * SIZE * "=" + "#"

def mirror():
    for line in list(range(1, SIZE + 1)) + list(range(SIZE, 0, -1)):
        print("|" + (-2 * line + 2 * SIZE) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 2 * SIZE) * " " + "|")

# main
bar()
mirror()
bar()
```