Python Basics III

Saif Ali

Department of Electrical Engineering

Jamia Millia Islamia

VARIABLES, EXPRESSIONS & STATEMENTS

Values and data types

A **value** is one of the fundamental things — like a letter or a number — that a program manipulates.

- 2 + 2
- "Welcome to Python!".

These values are classified into different **classes**, or **data types**

- 4 is an *integer*
- "Welcome to Python!" is a *string* so-called because it contains a string of letters.

type - function

If you are not sure what class/data type a value falls into, Python has a function called **type** which can tell you.

```
type("Welcome to JMI!")
```

```
str
```

```
type(17)
```

```
int
```

Source: http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html

Variables

A **variable** is a *name* that refers to a *value*.

One of the most powerful features of a programming language is the ability to manipulate **variables**.

The value is *assigned* to a name through the **assignment statement**

```
message = "Love conquers all."
n = 17
pi = 3.14159
```

Remember

The assignment statement binds a *name*, on the left-hand side of the operator, to a *value*, on the right-hand side.

Assignment Statement != Equals

The **assignment token**, `=`, should not be confused with *equals*, which uses the token `==`.

The assignment statement binds a *name*, on the left-hand side of the operator, to a *value*, on the right-hand side.

This is why you will get an error if you enter:

```
17 = n
SyntaxError: can't assign to literal
```

Tip

When reading or writing code, say to yourself “n is assigned 17” or “n gets the value 17”. Don’t say “n equals 17”.

State Table or State Snapshot

A table indicating all the variables in your program and the corresponding values to which they are bound.

```
message = "Love conquers all."
n = 17
pi = 3.14159
```

```
message → “Love conquers all.”
n → 17
pi → 3.14159
```

Remember

The state table is a map or a data representation. This type of table is used internally by the Python interpreter. It is not visible to the programmer anywhere.

Accessing or Referring to Variables

You can access a variable or refer to a variable simply by its name.

```
message
'Love conquers all.'
n
17
pi
3.14159
```

Variables are meant to vary!

The whole point of variables is that you can change their values as many times as you want in a program. They are meant to be *variable*!

```
day = "Thursday"
day
'Thursday'
day = "Friday"
day
'Friday'
day = 21
day
21
```

Variable Naming Rules

Variable names:

- can be arbitrarily long.
- can contain letters, digits and the underscore '_' character
- have to begin with a letter or an underscore.
- are case-sensitive
  - Bruce and bruce are different variable names
- the underscore character (_) can appear in a name. It is often used in names with multiple words, such as my_name or price_of_tea_in_china.

Tip

Although it is allowed, beginning variables names with an underscore is not recommended. Sometimes such variables have special meanings.
Eg: _system

Variable Naming Errors

```
1strank = "first"
SyntaxError: invalid syntax

more$ = 1000000
SyntaxError: invalid syntax

class = "Computer Science 101"
SyntaxError: invalid syntax
```

Keywords

Keywords are words that have pre-specified or special meanings.

Keywords define the language's syntax rules and structure, and they cannot be used as variable names.

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

Source: https://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html

Statements

A **statement** is an instruction that the Python interpreter can execute.

Quick Review

What kind of statement have we already looked at?

Operators and operands

Operators are special tokens that represent computations like addition, multiplication and division.

The values the operator uses are called **operands**.

```
20+32
hour-1
hour*60+minute
minute/60
5**2   (5+9)*(15-7)
```

Modulus operator

The **modulus operator** works on integers (and integer expressions) and gives the remainder when the first number is divided by the second. In Python, the modulus operator is a percent sign (%).

```
r  = 7 % 3
```

Integer division operator

The **integer division operator** works on integers (and integer expressions) and gives the quotient when the first number is divided by the second. In Python, the modulus operator is a percent sign (//).

```
q  = 7 // 3
r  = 7 % 3
```

Operator Precedence

When more than one operator appears in an expression, the order of evaluation depends on the **rules of precedence**. Python follows the same precedence rules for its mathematical operators that mathematics does. The acronym **PEDMAS** is a useful way to remember the order of operations:

1. **P**arentheses have the highest precedence
   a. force an expression to evaluate in the order you want.
   b. to make an expression easier to read, as in `(minute * 100) / 60`
2. **E**xponentiation has the next highest precedence, `2**1+1` is 3 and not 4, and `3*1**3` is 3 and not 27.
3. **M**ultiplication and both **D**ivision operators have the same precedence, which is higher than **A**ddition and **S**ubtraction, which also have the same precedence. So `2*3-1` yields 5 rather than 4, and `5-2*2` is 1, not 6.
4. Operators with the *same* precedence are evaluated from left-to-right.
   a. In algebra we say they are *left-associative*. So in the expression `6-3+2`, the subtraction happens first, yielding 3. We then add 2 to get the result 5.
   b. If the operations had been evaluated from right to left, the result would have been `6-(3+2)`, which is 1. (The acronym PEDMAS could mislead you to thinking that division has higher precedence than multiplication, and addition is done ahead of subtraction - don't be misled. Subtraction and addition are at the same precedence, and the left-to-right rule applies.)

Operations on strings

In general, you cannot perform mathematical operations on strings, even if the strings look like numbers. The following are illegal (assuming that message has type string):

```
message - 1          # Error
"Hello" / 123        # Error
message * "Hello"    # Error
"15" + 2             # Error
```

```
fruit = "banana"
baked_good = " nut bread"
print(fruit + baked_good)
```

Summary

- A **value** is one of the fundamental things — like a letter or a number — that a program manipulates.
- Each value has a type depending on the kind of data it is.
- The assignment statement assigns a value to a variable name.
- **Variables** are meant to vary, i.e, change in value.
- **Keywords** are reserved words that have a special meaning.
- **Statements** are instructions that the Python interpreter can execute.
- **Operators** and operands are used to specify computations and calculations.

Try it yourself
Open the "SA3-PR.ipnyb" Python notebook in Google Colab.

Link to Notebook

https://drive.google.com/file/d/1Es9gQmU7lrgip6QhV9iouD92GVcv4BOn/view?usp=sharing

https://shorturl.at/Zvlrf