Syntax: variables, expressions and statements

Lecture 02.01

By Marina Barsky

http://interactivepython.org/runestone/static/CS152f17/SimplePythonData/toctree.html

How to actually learn any new programming concept

Essential

Changing Stuff and Seeing What Happens

O RLY?

@ThePracticalDev

Trace the following code snippets

- Write and trace in visualizer: http://pythontutor.com/live.html#mode=edit

```
x = 5
y = 4
x = 2*y
```

```
x = 4
y = x + 2
x = y + 1
y = y + 6
```

```
a = 5
b = 15
```

Write code to swap which values *a* and *b* refer to: after your statements are executed, *a* should refer to the value that *b* used to refer to, and *b* should refer to the value that *a* used to refer to.
Hint: use a third variable.
Once you have written the code, trace your code manually using variable table

Language tokens (single words)

Reserved words

Values

Variables

Reserved Words

Reserved words have special meaning and used to give special instructions

`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`	

Fixed values: constants

Values have **type**

Numeric types

float

int

bool

```
>>> type(3.14)
<class 'float'>
```

```
>>> type(True)
<class 'bool'>
```

Sequence types

str

list

```
>>> type('writer')
<class 'str'>
```

```
>>> type([1,2,3])
<class 'list'>
```

Variables

- A variable is a named place in memory where we can store value and later retrieve it 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

Visualize programs with pythontutor.com

https://goo.gl/bcGWi8

Naming your variables: good names?

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

What is this program computing?

Naming your variables: better names?

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

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

What is this program computing?

Naming your variables

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

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

What is this program computing?

```
hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)
```

Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive
- Combine words using **snake_case**

```
Good:     spam     eggs    top_score   _speed
Bad:     23spam     #sign   var.12
All different:   spam    Spam    SPAM
```

Mnemonic Variable Names

Python is a weakly typed language

- When we declare new variable – the type is not declared
- The type is deduced (guessed) from the value
- We change the type of variable by assigning it a value of a different type

```
>>> x = 4
>>> type (x)
<class 'int'>
>>> x = 'abc'
>>> type (x)
<class 'str'>
```

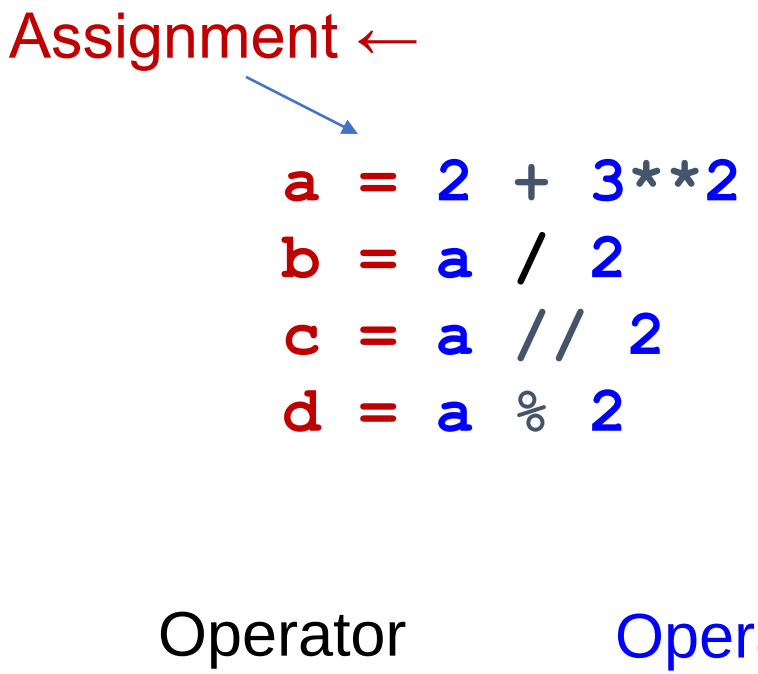
Expressions and assignments

Numeric expressions

String expressions

Assignment statement

Combining values and variables into expressions



= does not mean equal in Python, it means: **assign** value on the right into a variable on the left

Assigning expressions to variables

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right-hand side and a **variable to store the result**

```
x = 3.9 * x * ( 1 - x )
```

Numeric Expressions: try in IDLE

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> ll = jj // 5
4
>>> print(4 ** 3)
64
```

```
    4 R 3
  ------
5 | 23
    20
  ----
     3
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Int. division
**	Power
%	Remainder

Operator Precedence Rules

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”

Highest to lowest precedence:

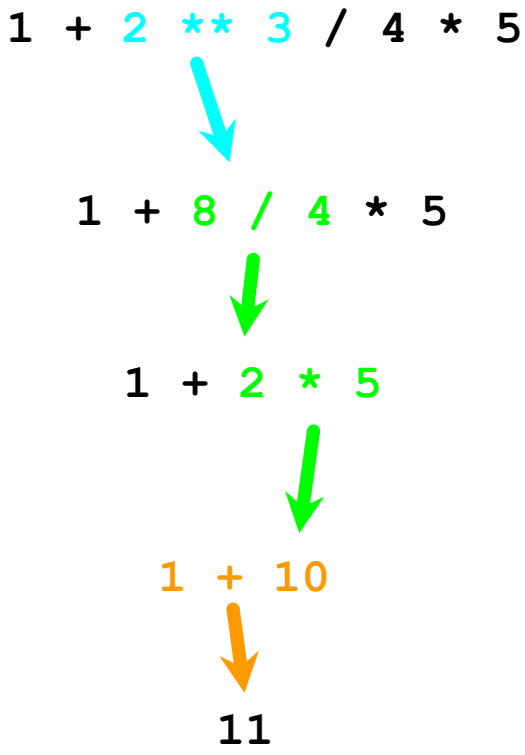
- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

Parenthesis
Power
Multiplication
Addition
Left to Right

Order of evaluation

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

Parenthesis
Power
Multiplication
Addition
Left to Right



The type of the result depends on the type of operands

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 **convertors** 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 and modulo operator

Division operator **/** always produces a floating point result

If you want an integer division (the whole part of the result) use operator **//**

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(9 // 2)
4
>>> print(99 // 100)
0
```

Integer division and modulo operator

If you want an **integer** division (the whole part of the result) use operator **//**

Modulo operator **%** produces the **remainder** of the integer division

```
>>> print(10 // 4)
2
>>> print(10 % 4)
2
```

String expressions

Values and variables of type str can also be combined into expressions

The meaning of the only valid two operators + and * is different for string operands:

- **+** concatenates strings
- * repeats strings

```
>>> 'first' + 'class'
'firstclass'
>>> 'bro' + 'ha' * 5
'brohahahahaha'
```

Operators cannot work on operands of two different types: number and string

You cannot "**add 1**" to a string

```
>>> 'hello' + 1
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in
<module>
    'hello' + 1
TypeError: must be str, not int
```

To concatenate strings with numbers we need to convert numbers to strings first - using *str()* convertor

```
>>> 'hello ' + str (1)
'hello 1'
```

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 "<pyshell#12>", line 1, in
<module>
    print (sval + 1)
TypeError: must be str, not int
>>> ival = int (sval)
>>> print (ival + 1)
124

>>> sval = 'Bob'
>>> ival = int (sval)
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in
<module>
    ival = int (sval)
ValueError: invalid literal for int()
with base 10: 'Bob'
```

Dialogue with users

input

print

User Input

We can instruct Python to pause and get data from the keyboard using the *input()* function

The *input()* function produces a string

```
name = input('Who are you? ')
print('Welcome', name)
```

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 ('Fahrenheit Temperature ? ')
fahr = float (inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print (cel)
```

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:

What happens if the user enters text instead of a number?

```
inp = input ('Fahrenheit Temperature ? ')
fahr = float (inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print (cel)
```

The try / except Structure

You surround a dangerous section of code with try and except

If the code in the try works - the except is skipped

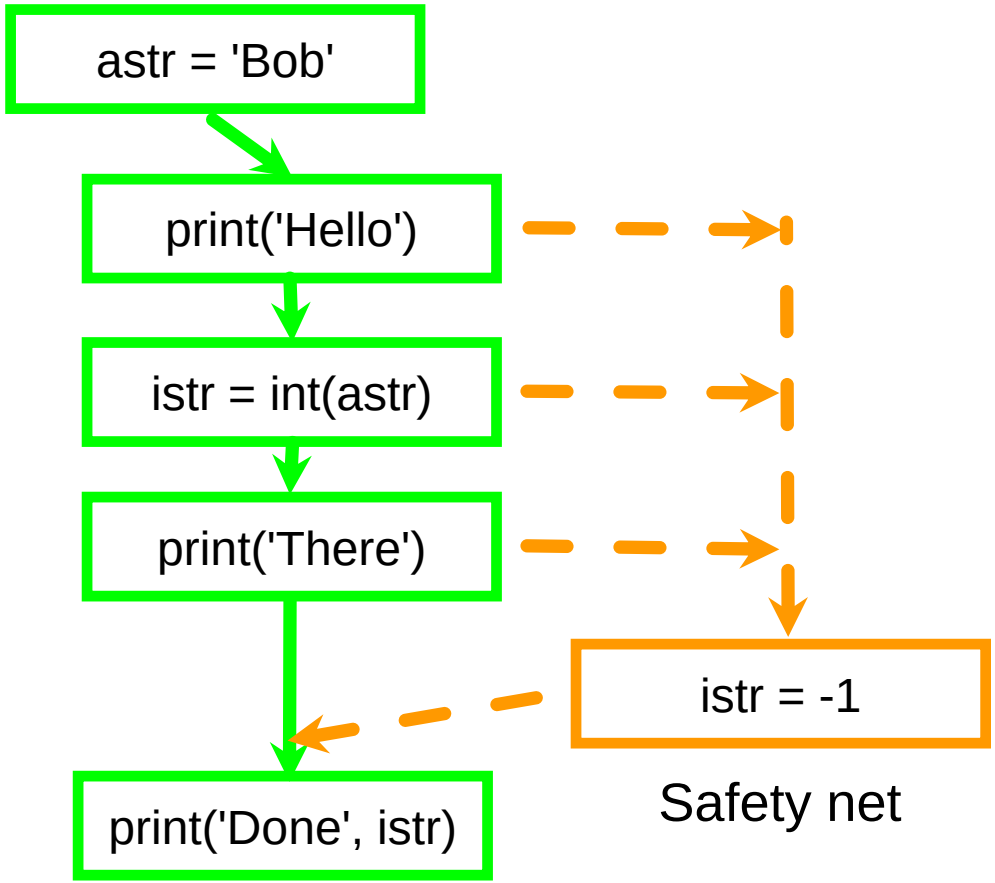
If the code in the try fails - it jumps to the except section

```
inp = input ('Fahrenheit Temperature ? ')
try:
    fahr = float (inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print (cel)
except:
    print ('Invalid input')
print ('this was Fahrenheit to Celsius converter')
```

try / except

```
astr = 'Bob'
try:
    print('Hello')
    istr = int(astr)
    print('There')
except:
    istr = -1

print('Done', istr)
```



Sample try / except

```
rawstr = input('Enter a number:')
try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print('Nice work')
else:
    print('Not a number')
```

```
>>Enter a number:42
Nice work
>>Enter a number:forty-two
Not a number
```