Programming for Data Science

Lab02 – Variables, Types, Operators and Expression

Instructor: Jakramate Bootkrajang
Based on Material by Kittipitch Kuptavanich

Outline

- Variables
- Data types
- Statement and Expression
- Operators
  - Operator precedence

Variables

- Variable in Python is a named object used for referencing data object
- Variables are created when first assigned, using the assignment operator (=)

```
pi = 3.14
radius = 11
area = pi * radius * radius
```

Variable names

- Must begin with a letter (a - z, A - B) or underscore (_)
- Other characters can be letters, numbers or _
- Case Sensitive
- Can be any (reasonable) length
- There are some reserved words which you cannot use as a variable name because Python uses them for other things.

Good variable names

- Choose meaningful name instead of short name. film_no is better than fn.
- Be consistent; film_no or FilmNo
- Begin a variable name with an underscore(_) character for a special case.
- For long variable names choose between two styles
  - this_is_really_long_name
  - thisIsReallyLongName

Python keywords

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

- You can list keywords with

Example of bad naming

```
a = 3.14159
b = 11.2
c = a * b * b
```

```
pi = 3.14159
diameter = 11.2
area = pi * diameter * diameter
```

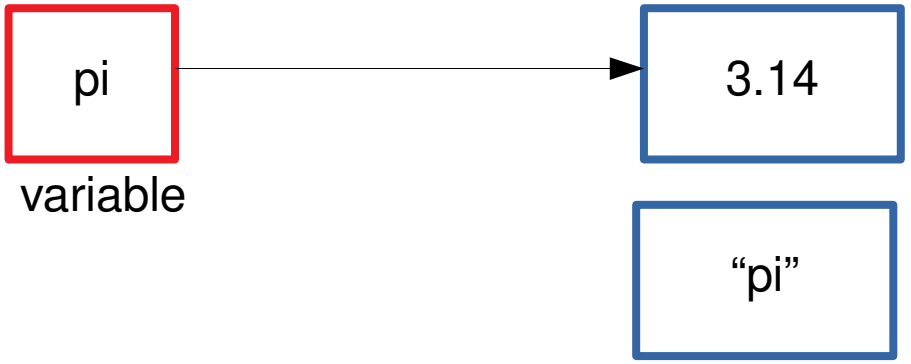
- Consider the above codes
- They are equivalent as viewed by Python interpreter
- However, the code on the right is ambigious
- Variable diameter should be radius ?

Just for fun

- There are competition for writing confusing codes.
- The most famous one being `The International Obfuscated C Code Contest’
- One example of winning entries is on the next page.

```
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#define p return
#define X typedef
#define E {
#define B }
#define stup(y)printf("%.11s\n",(char*)y);
#define $ printf("\x1b"
#define G(y)usleep(y);fflush(stdout);
#define O __DATE__+7
#define K printf(
X unsigned short P;
X int _;
 s=41456; //Sleeping time factor
X char H;
 o,b,a,f,g,k, m; // Oh, be a fine girl, kiss me
FILE *W;
X enum  {T,F}  z; //True & False
X double w;
 a5_, N = 11,A4=1<<3,a8_=17,A8=2*2,aI_=~-(__FLT_MAX_EXP__<<F);
H l[]="uDSYT8|TCPZ8|TVCTKp]X^EY|XKPC8aYTRUPp]ZPXU";H I[]="Fo~y*lcdn*yego*Oky~ox*ommy";/*
                                                    /\
                                                    c_
                                                   [9];
                         alpha Ursae Minoris A     main
                                                  */_ a1,
                                                 a3 ,a4=
                                                 024;P a2
                                                ,a5;w Z( w
                                               D,_ Y)E if(Y
                                               ==T)p F;if(Y
                                              ==!0)p D; w b=
                                             Z(D,Y/2);if(Y>2      /* alpha Ursae Minoris Ab*/
                                             )p b*b*D;p b*b ;
                                            B _ u(_ x) E if( x                              <2)p
                                            F;p x*u(x-unix); B
       /*alpha Ursae Minoris B*/           _ vir(H __[]) E _ e,
                                           q=0;for(e=T;e[__];e++)
          q=q*                             -~9+__[e]-48;p q;B P A
                                          (H *a7_,__a6_ );w V(_ v
                                         ,_ i){v%=0550;w D='-'-'-';
                                        for(i-=i;i^__CHAR_BIT__;i++
 )D+=Z(_POSIX_TRACE,i)*Z(0x1.1df46a226e211p-6 *v,2*i)/u(2*i);p D;B _ main(a6,n)_ a6;H**n;E _ a7,A1,a1_,a8,t,
    A6,D,Y;size_t a3_;_A2[]=E'q',W_OK,5911774,'y',13,1052160,77,15,1709568,'5',a8_,1314816,34,18,1446400
       ,91,22,1314816,12,28,F_OK B;time_t j =time(NULL);a4 *= ' ' ;float and [] =/*trolling is a art */
          E  46129548085432077390970880, 207.285645,  2143549501323367336505428844250293862400 B ;
            struct tm  R = *localtime (&j) ;for (;t=aI_,a5_[l];) l[a5_++] ^=-~__LINE__ ; A6=a4 +=
              R.tm_year;a4 /='d';ssize_t A3 =1;k=A6/'d',b=k/A8,a=k%4,o=(k-((k+A4)/25)+1)/F_TEST
                 ,m=(19*(A6%a4 )+k-b-o+__DBL_DIG__)%30,k=(A6%'d')/R_OK;H*a2_=NULL;f=(A6%'d')%
                    R_OK,g=(' '+2*a+SEEK_END*k-m-f)%7,k=__ATOMIC_SEQ_CST;W=fopen(__FILE__,
                      "r" ); $"[40m\033[2J\033[;H\x1b[?25l");if(*(H *)(_[])E X_OK B)stup
                        (and)A3 =getdelim(&a2_,&a3_,'\0', W);b=((A6%19)+N*m+22*g)/(N*
                          ')'),o=m+g-(~-A4)*b+'r',D=o/31;for(;a1[I];a1++,a5_=0xb3)
                             I[a1 ]^= '-'-'#';Y=o% ~-' ' +F;for(a3 =T;a3<A3 ;++
                                a3 )a2 =(a2  +a2_[a3 ])%255,a5 =(a5 +a2 )%t;
                                 if((a5 <<8|a2 )^s) goto a4_;$ "[1;12H  " )
                                ;G(a4 *s)a4 ^=a4 ;k*=2172211;while(a5_-->-1)
                                E $ "[40m\x1b[2J");for(a6=W_OK;a6<025;a6 +=3)
                               E a7  = A2 [ a6 -__FLT_RADIX__ ] - 0102, a8 =k
                              + a6 [A2],a1 = A2[~-a6 ] *0x1.db6db6db6a92ap+0 ;
                              a1-=~-__DBL_DIG__,a1_=a7 *V(a5_,_XOPEN_VERSION ) -
                              a1*V(90-a5_,a7 )+66,A1 =(a7*V(90-a5_,a1 ) +a1 *V (
                             a5_,a8 )+0xe)/1.8571428571;if(A1 >T&&a1_>0) E $"[3"
                             "8;2;%d;%d;%dm", (a8>> 16  )&t,(a8>>8)& 255,a8 &t);$
                            "[%d;%dH%c %.*s",A1,a1_         ,a5_>90?46:111,6,a5_>45
                            ?"":l+ ((a6-F_TLOCK) /              __ATOMIC_RELEASE]*6);
                            B B G(u(A4 ) )B/**/                 a4_:$    "[H\x1b[0m"
                           "\x1b[?25h") ;if(                       ( D^ (R.tm_mon+1))
                           == (Y-= R./* */                           tm_mday) ) E   K
                          "%s\n",I);B K                                 ""     "%d\n",
                         vir(O)  );p                                      a4 ;B //END\
                         OF  THIS                                           ENTRY IS\
                        N E A R                                               SO NEAR\
                       .*****                                                   ******\
                       ****                                                       ***\
```

Types of Data Objects



- Variable can point to any of data objects (but not at the same time).
  - This is very convenient for Python programmer
- We can use a function `type()’ to investigate object type.

Some basic data types

- int – Integer 1,3,5,9,-5,-100
- float – Floating point number 2.2,500.1,-10.5
- complex – Complex number 1+2j
- str – String, list of character “abc”, ‘def’, “”
- bool – True, False
- list – [1,”good”, True]

Some Built-in Types

```
01 # Some Builtin Types
02 import math
...
05 def f():
06     print("This is a user-defined function")
07
08 print("Some basic types in Python:")
09 print(type(2))           # int
10 print(type(2 ** 500))    # int
11 print(type(2.2))         # float
12 print(type("2.2"))       # str  (string)
13 print(type(2 < 2.2))     # bool (boolean)
14 print(type(math))        # module
15 print(type(math.tan))    # builtin_function_or_method
16 print(type(f))           # function (user-defined function)
17 print(type(type(42)))    # type
```

Some Built-in Types [2]

```
23 print("Later in the course...")
24 print(type(Exception()))    # Exception
25 print(type(range(5)))       # range
26 print(type([1, 2, 3]))      # list
27 print(type((1, 2, 3)))      # tuple
28 print(type({1, 2}))         # set
29 print(type({1: 42}))        # dict (dictionary or map)
30 print(type(2 + 3j))         # complex  (complex number)
31
32 # Some Builtin Constants
33 print("Some builtin constants:")
34 print(True)
35 print(False)
36 print(None)
```

Dynamic typing

- Assigning new value of different type to existing variable

```
>>> theSum = 0
>>> theSum
0
>>> theSum = theSum + 1
>>> theSum
1
>>> theSum = True
>>> theSum
True
```

Figure 1.3: Variables Hold References to Data Objects

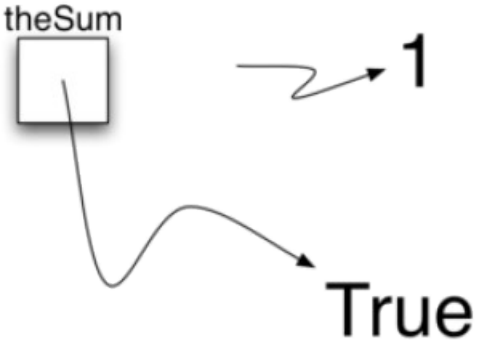


Figure 1.4: Assignment changes the Reference

Garbage collection

- Unreferenced data object becomes garbage
- Garbages will be collected by Python automatically

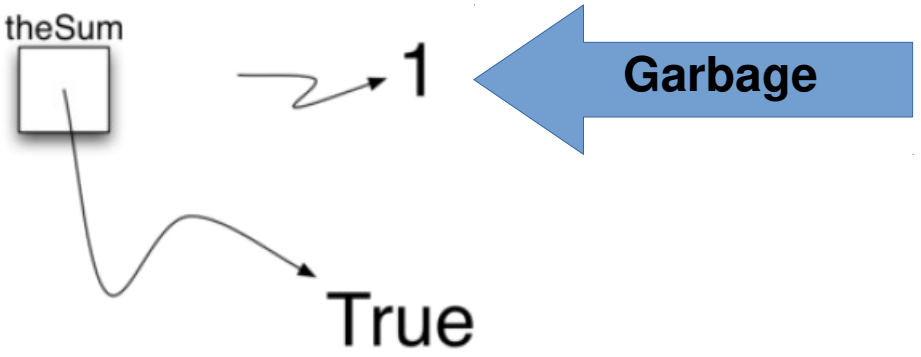


Figure 1.4: Assignment changes the Reference

Expression and Statement

- An expression is a combination of values, variables, and operators.
  - An expression can be evaluated to a value
- A statement is a unit of code that the Python interpreter can execute.
- Expression has value; a statement does not.

```
>>> x = 3     # statement
>>> x == 3    # expression
True
>>> x           # expression
3
```

Operators

- There are 5 groups of operators in Python
  - Arithmetic operators
  - Relational operators
  - Bitwise operators
  - Assignment operators
  - Logical operators

List of Operators

Category	Operators
Arithmetic	+, -, *, /, //, **, %, - (unary), + (unary)
Relational	<, <=, >=, >, ==, !=,
Bitwise	<<, >>, &, |, ^, ~
Assignment	+=, -=, *=, /=, //=, **=, %=, <<=, >>=, &=, |=, ^=
Logical	and , or, not

Note

- `/` is normal division `3 / 2 == 1.5`
- `//` is *floored* division `3 // 2 == 1`
  `-3 // 2 == -2`
- `**` is power

Data Types Affect Semantics

```
>>> print(3 * 2)            # normal multiplication
6

>>> print(3 * "abc")        # string repetition
abcabcabc

>>> print(3 + 2)            # normal addition
5

>>> print("abc" + "def")    # for concatenating strings
abcdef

>>> print(3 + "def")
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

Arithmetic Operator Precedence

- Follows PEMDAS rule
  - **P**arentheses
  - **E**xponentiation
  - **M**ultiplication and **D**ivision
  - **A**ddition and **S**ubtraction
- Operators with same precedence evaluated from left to right
- Let's calculate 2^{3^5}

Relational Operators and Boolean Expression

- Boolean Expression is an expression which evaluated to truth value (either True or False)

```
>>> 5 == 5
True
>>> 5 == 6
False
```

- True or False is not a string but it is a data object of type bool

Boolean Expressions [2]

- Operator (==) is called a relational operator
- There are other relational operators

```
x != y      # x is not equal to y
x > y       # x is greater than y
x < y       # x is less than y
x >= y      # x is greater than or equal to y
x <= y      # x is less than or equal to y
```

Boolean Expression [3]

- Be careful with floating-point number comparisons

```
>>> print(0.1 + 0.1 == 0.2)
True                  # True, but...

>>> print(0.1 + 0.1 + 0.1 == 0.3)
False

>>> print(0.1 + 0.1 + 0.1)
0.3                   # seems ok

>>> print((0.1 + 0.1 + 0.1) - 0.3)
5.55111512313e-17     # (tiny, but non-zero!)
```

Better way to do comparison with Float

```
09 d1 = 0.1 + 0.1 + 0.1
10 d2 = 0.3
11 print(d1 == d2)           # still False, of course
12 epsilon = 10 ** -10
13 print(abs(d2 - d1) < epsilon)  # True!
14
15 # Once again, using an almostEqual function
16 # (that we will write)
17 def almostEqual(d1, d2, epsilon=10 ** -10):
18     return (abs(d2 - d1) < epsilon)
19
20 d1 = 0.1 + 0.1 + 0.1
21 d2 = 0.3
22 print(d1 == d2)            # still False, of course
23 # True, and now packaged in a reusable function!
24 print(almostEqual(d1, d2))
```

Bitwise operators

- Bitwise operation is to perform and (&), or (|), exclusive or (^) and not (~) on number(s) bit by bit
- Example

```
  0110          0110          0110
& 1100        | 1100        ^ 1100        ~ 1100
------        ------        ------        ------
  0100          1110          1010          0011
```

ทำ Operation & ในแต่ละหลักแยกกัน

Bitwise “~” operators

- “~” is used to reverse bit
- Python uses two’s complement number representation
  - Positive number Most Significant Bit (MSB) bit is 0. Example, 3 is stored as 011
  - Negative number MSB is 1
- So executing ~3 in Python is like
  - ~[011] → [100] → which is -4 in two’s complement notion.
  - What’s [1010] in decimal ?

Bitwise “~” operators [2]

- Let’s do it for negative number -35
  - 35 = [0100011]
  - -35 = [1011100] + [0000001] = [1011101]
  - ~(-35) = [0100010] = 34
- In general, ~x is equal to -x-1

```
>>> ~-35
34
>>> int('0100010', 2)  # convert number in base 2
34
```

Exercise: Bitwise Operations

```
# 0 1 1 0 = 4 + 2 = 6
# 0 1 0 1 = 4 + 1 = 5
>>> print("6 & 5 =", (6 & 5))
6 & 5 = _____
>>> print("6 | 5 =", (6 | 5))
6 | 5 = _____
>>> print("6 ^ 5 =", (6 ^ 5))
6 ^ 5 = _____
>>> print("6 << 1 =", (6 << 1))
6 << 1 = _____
>>> print("6 << 2 =", (6 << 2))
6 << 2 = _____
>>> print("6 >> 1 =", (6 >> 1))
6 >> 1 = _____
```

Shift operations

- You can shift bit representation of number in Python to the left or to the right using
  - left shift (<<) and
  - right shift (>>) operator
- Left shifting is to shift bit representation of x to the left 2 positions and to fill empty bits with zeros
  - 0010 << 2
  - 10__ → 1000

Shift operations

- For right shift, the left-most bit will be used to fill the empty bits to preserve the sign

Operation	Values	
Argument x	[01100011]	[10010101]
x << 4	[0011*0000*]	[0101*0000*]
x >> 4 (arithmetic)	[*0000*0110]	[*1111*1001]

- Exercise
  - (x << n) equals to _____
  - (x >> n) equals to _____

Logical Operators

- There are 3 types of logical operators in Python – and, or, not
- Example
  - x > 0 and x < 10 evaluates to True when x is greater than 0 and x is less than 10
  - n % 2 == 0 or n % 3 == 0 evaluates to True when either one is true
- Usually operands of logical operator should be boolean expression but non-zero number is treated as True

```
>>> 17 and True
True
```

Short-Circuit Evaluation

- Python stops evaluating long boolean expression once the result becomes 100% clear
- For example
  - Expr1 and Expr2 and Expr3
    - If Expr1 is False the rest is not needed because the whole thing will be False

Short-Circuit Evaluation [2]

```
>>> x = 1
>>> y = 0
>>> print((y != 0) and ((x / y) != 0))  # Works!
False
>>> print(((x / y) != 0) and (y != 0))  # Crashes!
...
ZeroDivisionError: division by zero

>>> print((y == 0) or ((x / y) == 0))   # Works!
True
>>> print(((x / y) == 0) or (y == 0))   # Crashes!
...
ZeroDivisionError: division by zero
```

Short-Circuit Evaluation [3]

```
25 def isPositive(n):
26     result = (n > 0)
27     print("isPositive(", n, ") =", result)
28     return result
29
30
31 def isEven(n):
32     result = (n % 2 == 0)
33     print("isEven(", n, ") =", result)
34     return result
35
36 print("Test 1: isEven(-4) and isPositive(-4)")
37 print(isEven(-4) and isPositive(-4))  # Calls both
38 print("----------")
39 print("Test 2: isEven(-3) and isPositive(-3)")
40 print(isEven(-3) and isPositive(-3))  # Calls only one
```

Truth Value Testing

- In Python every object is associated with “truth value”
- The following is evaluated to False
  - `None`
  - `False`
  - zero of any numeric type, for example, `0, 0.0, 0j.`
  - any empty sequence, for example, `'', (), [].`
  - any empty mapping, for example, `{}`.

Assignment operators

- For assigning values to variables

Assignment operators in Python

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5

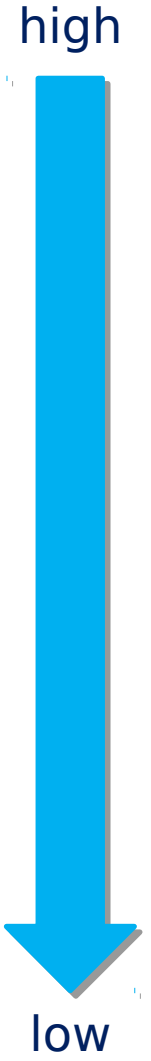
Assignment operators [2]

**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

Operator	Description
(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or tuple display, list display, dictionary display, set display
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
**	Exponentiation
+x, -x, ~x	Positive, negative, bitwise NOT
*, /, //, %	Multiplication, division, remainder
+, -	Addition and subtraction
<<, >>	Shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
not x	Boolean NOT
and	Boolean AND
or	Boolean OR
if – else	Conditional expression
lambda	Lambda expression

Arithmetic Operators

Boolean Operators



References

- **https://docs.python.org/3/library/stdtypes.html**
- **https://docs. Python.org/3.4/reference/expressions.html**
- **https://docs. Python.org/3.4/tutorial/inputoutput.html**
- **https://docs. Python.org/3.4/library/stdtypes.html#old-string-formatting**
- **http://www.cs.cmu.edu/~112/notes/notes-data-and-exprs.html**
- **Miller, B., and Ranum, D. *Problem Solving with Algorithms and Data Structures Using Python*,**
- **Guttag, John V. *Introduction to Computation and Programming Using Python, Revised***