香港浸會大學
HONG KONG BAPTIST UNIVERSITY

Python Basics

JOUR7280

Big Data Analytics for Media and Communication

Instructor: Dr. Xiaoyi Fu

Outline

- Python program flow
- Variables, expressions, and statements
- Unser inputs

Syntax Error

- You’re talking to python and this is a language that you don't already know.
- The computer is going to seem to judge your mistakes harshly.
- Syntax error doesn't mean that Python thinks you're bad.
- Syntax error means Python is lost.
  - It just doesn't have really friendly words when it says it's lost
  - Your syntax is not something that Python understands.

Assignment Statement

Program	Output
`x = 1`	1
`print(x)`	
`x = x+1`	2
`print(x)`	

Take whatever's in x, which is a 1, and then add 1 to it, which becomes 2, and then stick it back in x.

Reserved Words

- Words that have very special meaning to Python.
  - They have one and only one meaning to Python.
  - Use it the way Python expects us to use it
- Can NOT use reserved words as variable names / identifiers

```
and       del       global    not       with
as        elif      if        or        yield
assert    else      import    pass
break     except    in        raise
class     finally   is        return
continue  for       lambda    try
def       from      nonlocal  while
```

Sentences or Lines

```
x = 2
x = x + 2
print(x)
```

← Assignment statement

← Assignment with expression

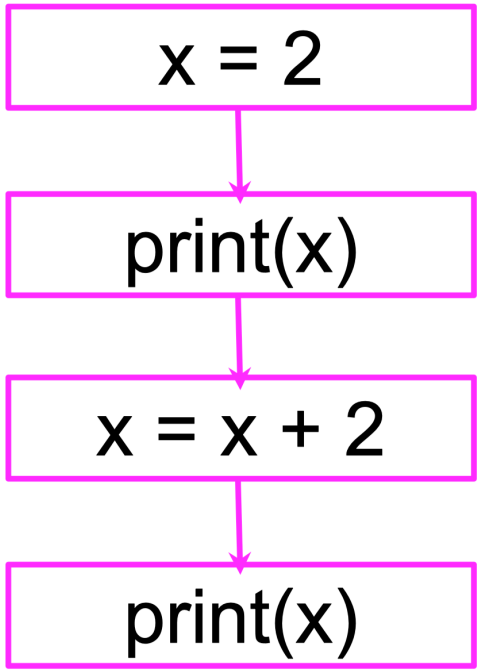
← Print function

Variable Operator Constant Function

Program Steps or Program Flow

- Like a recipe or installation instructions, a program is a sequence of steps to be done in order.
- Some steps are conditional – they may be skipped.
- Sometimes a step or a group of steps is to be repeated.
- Sometimes we store a set of steps to be used over and over as needed several places throughout the program.

Program flow: Sequential steps



Program

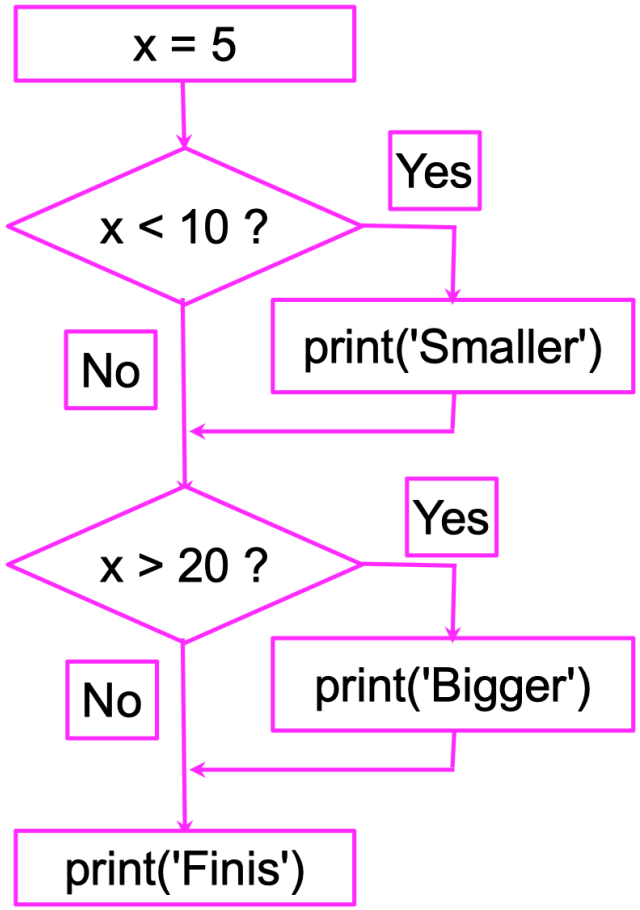
```
x = 2
print(x)
x = x + 2
print(x)
```

Output

```
2
4
```

- When a program is running, it flows from one step to the next. As programmers, we set up “paths” for the program to follow.

Program flow: Conditional steps



Conditional statement

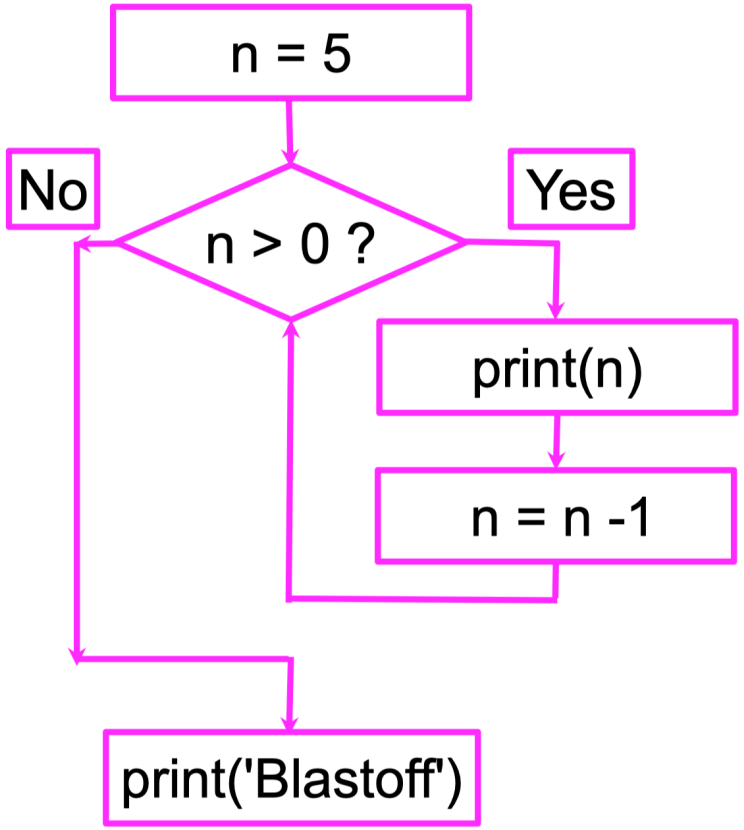
Program

```
x = 5
if x < 10:
    print('Smaller')
if x > 20:
    print('Bigger')
print('Finish')
```

Output

Smaller
Finish

Program flow: Repeated steps



Program

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Blastoff!')
```

Output

5
4
3
2
1
Blastoff!

Loops (repeated steps) have iteration variables that change each time through a loop.

Constants

- Fixed values such as numbers, letters, and strings
- Are called “constants” because their value does NOT change
- Numeric constants are as you expected
- String constants use single quotes (‘) or double quotes (")

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”
- A variable is a name that refers to a value.
- 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
```

Python variable naming rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive
- Cannot use reserved words
- Good spam eggs spam23 _speed
- Bad 23spam #sign var.12
- Different spam Spam SPAM

Sentences or Lines

```
x = 2
x = x + 2
print(x)
```

← Assignment statement

← Assignment with expression

← Print function

Variable Operator Constant Function

Choosing mnemonic variable names

```
x1q3z9ahd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ahd * x1q3z9afd
print(x1q3p9afd)
```

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

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

What is this bit of code doing?

- Python interpreter sees all three of these programs as exactly the same
- Humans will most quickly understand the intent of the third program

- $2\ expression.ipynb$

Assignment Statement

- 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)
```

Expression

- The right side is an expression
- The reason why it's possible to have the same variable on both sides because right-hand side happens first, ignoring left-hand side.
- Once the expression is evaluated, the result is placed in (assigned to) x

Numeric expressions

- **Operators** are special symbols that represent computations like addition and multiplication.
- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different than in math.
- $2\ expression.ipynb$

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

- $2\ expression.ipynb$

Order of Evaluation

- When we string operators together
  - Python must know which one to do first
- This is called operator precedence
- Which operator "takes precedence" over the others?

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

Operator Precedence Rules

- Highest precedence rule to lowest precedence rule:
  - Parentheses are always respected
  - Exponentiation (raise to a power)
  - Multiplication, division & remainder
  - Addition, subtraction
  - Left to right

Parentheses

Power

Multiplication

Addition

Left to Right

Operator Precedence

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

- Output: 11.0

Parentheses

Power

Multiplication

Addition

Left to Right

Operator Precedence

- Remember the rules top to bottom
- When writing code
  - Use parentheses
  - Keep mathematical expressions simple enough so that they are easy to understand
- Break long series of mathematical operations up to make them more clear

Parentheses

Power

Multiplication

Addition

Left to Right

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
  - Concatenate: put together

```
ddd = 1+4
print(ddd)

eee = ‘hello '+'there'
print(eee)
```

$2\ expression.ipynb$

“Simple” data types

Type	Example	Description
int	x = 1	integers (i.e., whole numbers)
float	x = 1.0	floating-point numbers (i.e., real numbers)
complex	x = 1 + 2j	Complex numbers (i.e., numbers with real and imaginary part)
bool	x = True	Boolean: True/False values
str	X = “abc”	String: characters or text. A string is a sequence of characters.
NoneType	X = None	Special object indicating nulls

Type Matters

- Python knows what "type" everything is
- Some operations are prohibited
- E.g., you cannot "add 1" to a string
- We can ask what type something is by using `type()` function

```
In [7]: eee = 'hello '+'there'
        eee = eee+1

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-af408901484c> in <module>
      1 eee = 'hello '+'there'
----> 2 eee = eee+1

TypeError: can only concatenate str (not "int") to str

In [9]: type(eee)

Out[9]: str

In [10]: type('hello')

Out[10]: str

In [11]: type(1)

Out[11]: int
```

2 expression.ipynb

Type Conversions

- When you put an integer and a floating point number in one expression, the integer is implicitly converted to a float
- You can control this with the built-in functions int() and float()

```
In [3]:  j = 5.9
         k = int(j)
         print(k)
         type(k)

         5

Out[3]:  int
```

```
In [12]: x = 100 + 1.0
         print(x)

         101.0

In [15]: print(100+float(1))

         101.0

In [13]: i = 42
         type(i)

Out[13]: int

In [14]: f = float(i)
         print(f)
         type(f)

         42.0

Out[14]: float
```

2 expression.ipynb

Integer division

- Integer division produces a floating point result

```
In [16]: print(10/2)
         5.0

In [17]: print(9/2)
         4.5

In [18]: print(99/100)
         0.99

In [19]: print(10.0/2.0)
         5.0
```

$2\ expression.ipynb$

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

```
In [20]: sval = '123'
         type(sval)

Out[20]: str

In [21]: print(sval+1)

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-21-d31b14f87b22> in <module>
         ----> 1 print(sval+1)

         TypeError: can only concatenate str (not "int") to str
```

```
In [22]: ival = int(sval)
         type(ival)

Out[22]: int

In [23]: print(ival+1)

         124

In [24]: nsv = 'hello world'
         niv = int(nsv)

         ---------------------------------------------------------------------------
         ValueError                                Traceback (most recent call last)
         <ipython-input-24-7b19be68013f> in <module>
               1 nsv = 'hello world'
         ----> 2 niv = int(nsv)

         ValueError: invalid literal for int() with base 10: 'hello world'
```

User Input

- We can instruct Python to pause and read data from the user using the input() function
  - The parameter to the input function is what's called a prompt
- The input() function returns a string
- We can also input a file (more on file inputs later)

```
In [*]: name = input('Who are u?')
        print('Welcome', name)

        Who are u?
```

```
In [27]: name = input('Who are u?')
         print('Welcome', name)

         Who are u?xiaoyi
         Welcome xiaoyi
```

$2\ expression.ipynb$

Comments in Python

- Anything after a # is ignored by Python
- Why comments?
  - Describe what is going to happen in a sequence of code
  - Document who wrote the code or other ancillary information
  - Turn off a line of code – perhaps temporarily

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.
- Later we will deal with bad input data.

```
# convert elevator floors
inp = input('Europe floor?')
usf = int(inp)+1
print('US Floor', usf)
```

Europe floor? 0

US Floor 1

$2\ expression.ipynb$

Thank You