香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF
DATA SCIENCE
數據科學學院

MDS 5111 Python Programming

Lecture 1 Course Introduction

Tongxin Li
School of Data Science
The Chinese University of Hong Kong, Shenzhen

About Me

Research Interests: Control Systems, AI+Energy, Reinforcement Learning

Previous

The Chinese University of Hong Kong Mathematics & Information Engineering

2017

CMS @ Caltech

2020

Interned @ AWS

2021

Interned @ AWS (return offer)

2022

ACM SIGEnergy Doctoral Dissertation Award (Honorable Mention)

Assistant Professor, Presidential Young Fellow

School of Data Science

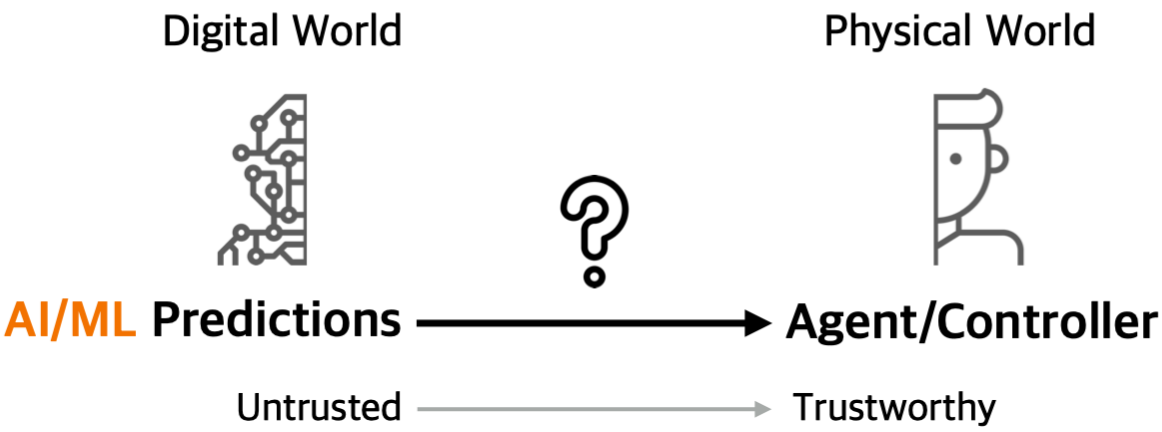
The Chinese University of Hong Kong, Shenzhen

Present

Adjunct Assistant Professor

Shenzhen Loop Area Institute

Research Overview



Research Overview

Theory

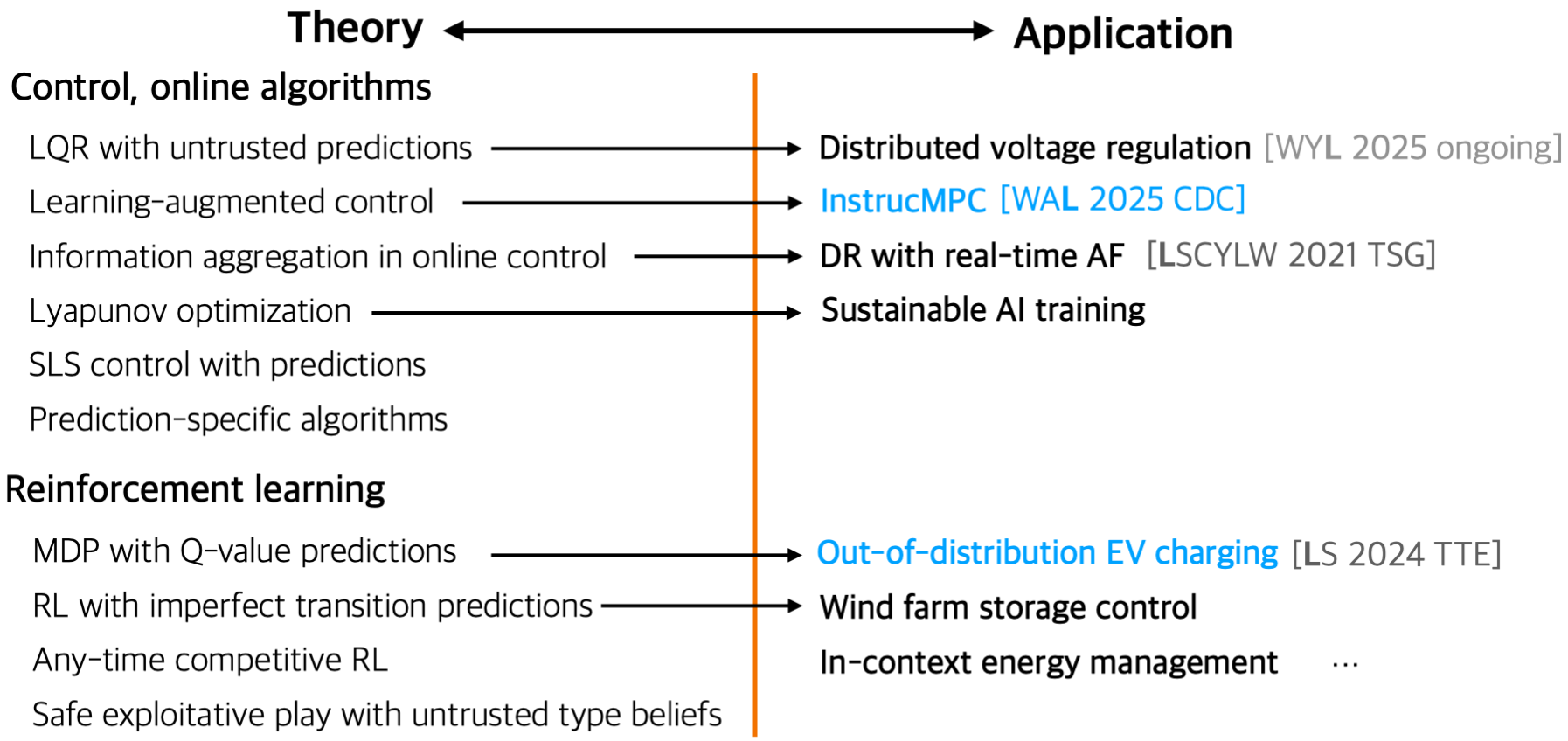
Control, online algorithms

LQR with untrusted predictions	[LYQSYWL 2022 SIGMETRICS] [LLY 2024 NeurIPS]
Learning-augmented control	[L 2025 submitted to TAC]
Information aggregation in online control	[LCSWL 2021 SIGMETRICS]
Bounded-regret model predictive control	[LHQLW 2022 NeurIPS]
System-level networked control with predictions	[WYL 2025 CDC, submitted to TAC]
Prediction-specific learning-augmented algorithms	[LCL 2026 SIGMETRICS]

Reinforcement learning

MDP with Q-value predictions	[LLRW 2023 NeurIPS]
Safe exploitative play with untrusted type beliefs	[LHRW 2024 NeurIPS]
Any-time competitive RL	[YLLWR 2023 NeurIPS]
RL with imperfect transition predictions	[LCLWW 2025 NeurIPS Spotlight]

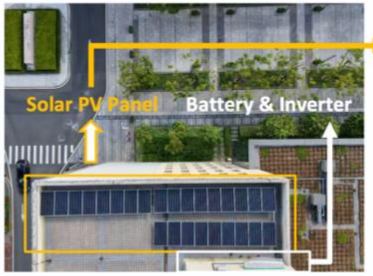
Research Overview & Plan

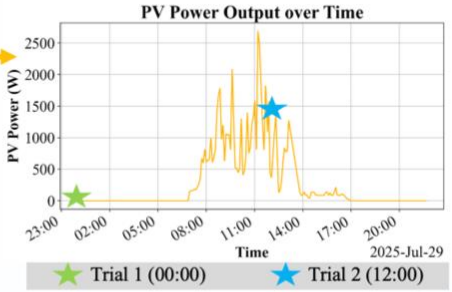


Sustainable AI Infrastructure

PV Modeling [Accepted PESIM 2026]

Sustainable AI Training [submitted to SIGMETRICS 2026]





Homepage:

https://tongxin.me/

MDS 5111:

https://tongxin.me/MDS5111-2026spring

Teaching Assistants:

Tinko Bartels

Ruixiang Wu

Yu Mao

Learning Objectives

- This course introduces data science using Python
- Students will learn data analysis and visualization using data structures and libraries in Python
- Students will learn to apply tools to solve real data analysis problems

Assessment

2 Mini-Projects	15%*2
Mid-term exam	30%
Final exam	40%

Course Materials

- All lecture notes and sample code used in classes will be provided to students via Blackboard
- Readings
  - Required
    - Wes McKinney, "Python for data analysis", 2012, download link: https://bedford-computing.co.uk/learning/wp-content/uploads/2015/10/Python-for-Data-Analysis.pdf
  - Recommended
    - Wes McKinney, "Python for data analysis: Data Wrangling with Pandas, NumPy, and IPython, 2nd Edition", 2019, download link: https://www.programmer-books.com/wp-content/uploads/2019/04/Python-for-Data-Analysis-2nd-Edition.pdf
    - David Amos, Dan Bader, Joanna Jablonski, Fletcher Heisler, "Python Basics: A Practical Introduction to Python 3", 2020, download link https://static.realpython.com/python-basics-sample-chapters.pdf

Course Components

Activity	Hours/week
Lecture	3× 14
Tutorial	No

Tentative Teaching Plans

Week	Content/ topic/ activity
1	Course Introduction & Brief Introduction to Python
2	Object Oriented Programming
3	Numpy Basics
4	Pandas I
5	Pandas II
6	Data Loading and File System
7	Data Wrangling
8	Mid-term Exam
9	Matplotlib for Visualization
10	Data Aggregation
11	Machine Learning Basics
12	Linear Regression and Classification
13	Data Analysis with Pandas
14	Review for final exam

Programmer

- Professional programmer writes computer programs and develops software
- A junior programmer gets high salary in an INTERNET company like Tencent
- A programmer can earn up to 500k – 1m USD in Google
- Software and INTERNET are huge industries

Google

Alibaba.com

Tencent

Microsoft

Meta

Programmer

- Professional programmer writes computer programs and develops software

- ~~A junior programmer gets high salary in an INTERNET company like Tencent~~

—

- ~~A programmer can earn up to 500k – 1m USD in Google~~

- Software and INTERNET are huge industries

美国CS就业梦碎！狂投5000家0 Offer，名校毕业00后被麦当劳惨拒

新智元 新智元 2025年08月14日 19:02 北京

117人 ☆星标

新智元报道

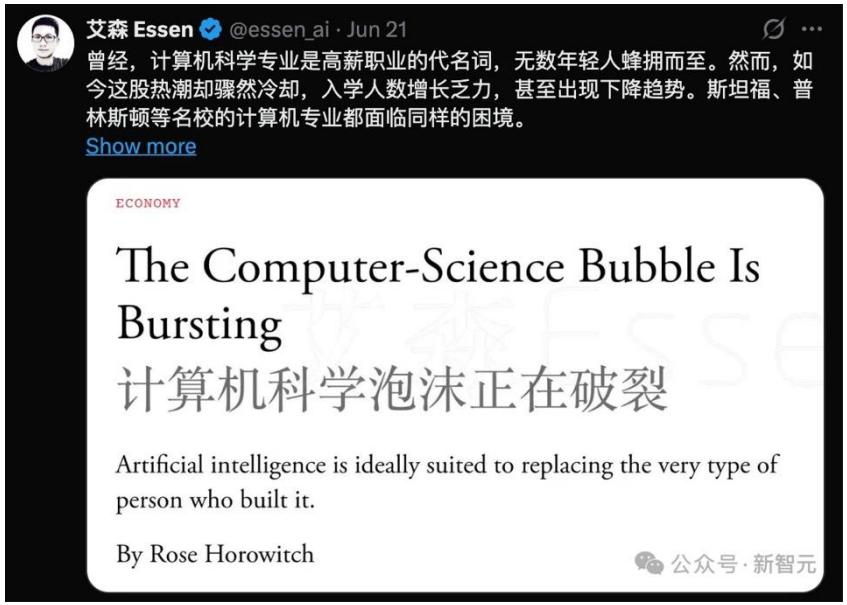
编辑：英智

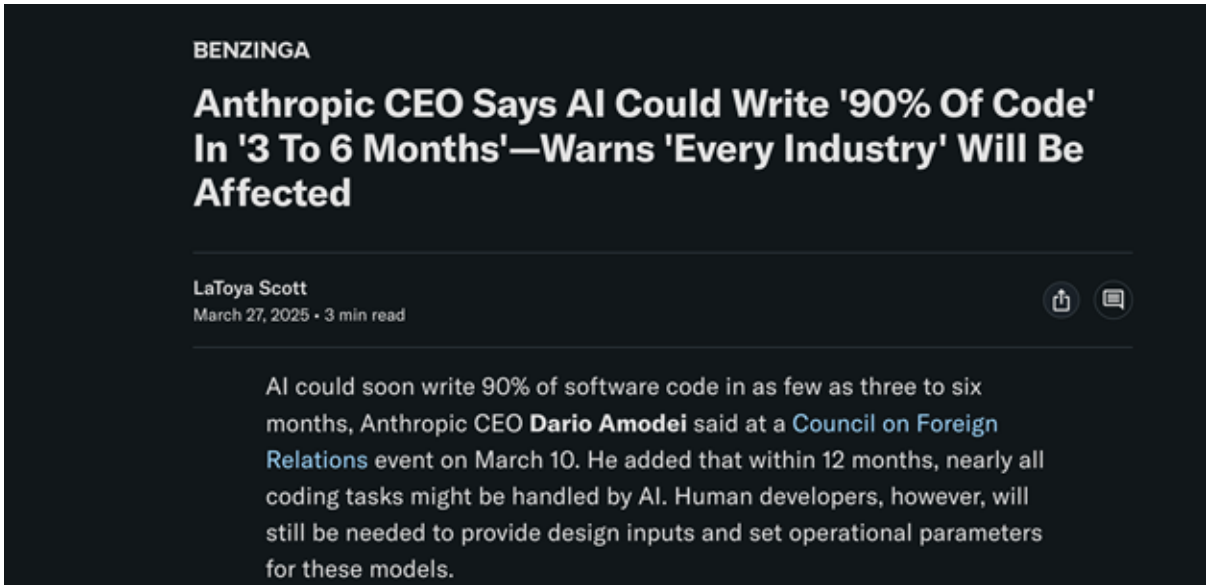
【新智元导读】当手握名校计算机学位的毕业生，发现唯一向他招手的竟是快餐店时，一个时代的神话或许已悄然落幕。科技巨头曾许诺的黄金饭碗为何被AI无情砸碎？本文带你倾听亲历者的心声。

毕业即失业，如今，却成为了CS的「铁律」。

Goodbye, $165,000 Tech Jobs. Student Coders Seek Work at Chipotle.

As companies like Amazon and Microsoft lay off workers and embrace A.I. coding tools, computer science graduates say they're struggling to land tech jobs.





BENZINGA

Anthropic CEO Says AI Could Write '90% Of Code' In '3 To 6 Months'—Warns 'Every Industry' Will Be Affected

LaToya Scott
March 27, 2025 • 3 min read

AI could soon write 90% of software code in as few as three to six months, Anthropic CEO **Dario Amodei** said at a Council on Foreign Relations event on March 10. He added that within 12 months, nearly all coding tasks might be handled by AI. Human developers, however, will still be needed to provide design inputs and set operational parameters for these models.

TECH

Satya Nadella says as much as 30% of Microsoft code is written by AI

PUBLISHED TUE, APR 29 2025•9:33 PM EDT | UPDATED TUE, APR 29 2025•9:58 PM EDT

Jordan Novet
@JORDANNOVET

Jonathan Vanian
@IN/JONATHAN-VANIAN-B704432/

SHARE

KEY POINTS

- Microsoft CEO Satya Nadella on Tuesday said that as much as 30% of the company's code is now written by artificial intelligence.

What Should We Do?

The AI Revolution: Our Generation's 'Dot-Com' Moment

Just like the internet boom of the early 2000s, AI is creating a new frontier of opportunity. The landscape is changing, but the core skills for success are more important than ever.

Why Coding is Still Your Key in the Age of AI?

Aim Higher: AI develops code. You develop the AI.

Get Hired: Prove your problem-solving skills in technical interviews.

Work Smarter: Need knowledge to prompt and debug for more effective vibe coding.

Learning Outcome

In the age of AI …

- Be able to write, compile and execute Python programs, including making use of Python's object-oriented methodology
- Be able to make use of some basic data structures and libraries to do some basic data analysis, as well as to conduct predictive model design
- Be able to use Matplotlib to achieve basic data visualizations
- Be able to comprehensively think and apply appropriate tools to solve some real data analysis problems

Python Basics

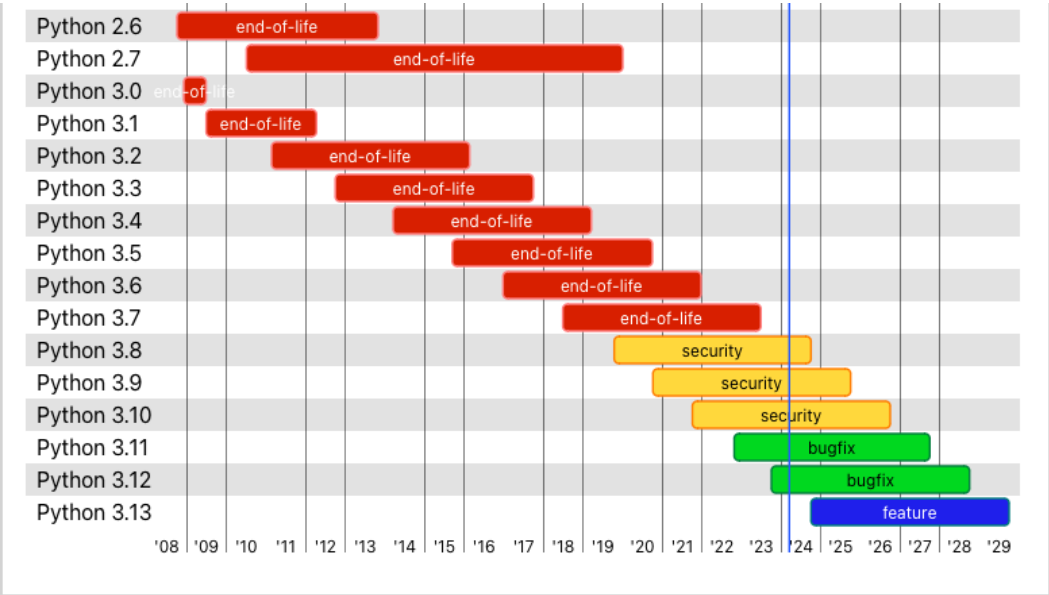
Introduction to Python

- Developed by Guido van Rossum in 1989, and formally released in 1991
- An open source, object-oriented programming language
- Powerful libraries
- Powerful interfaces to integrate other programming languages (C/C++, Java, and many other languages)
- Programming language of the year 2010

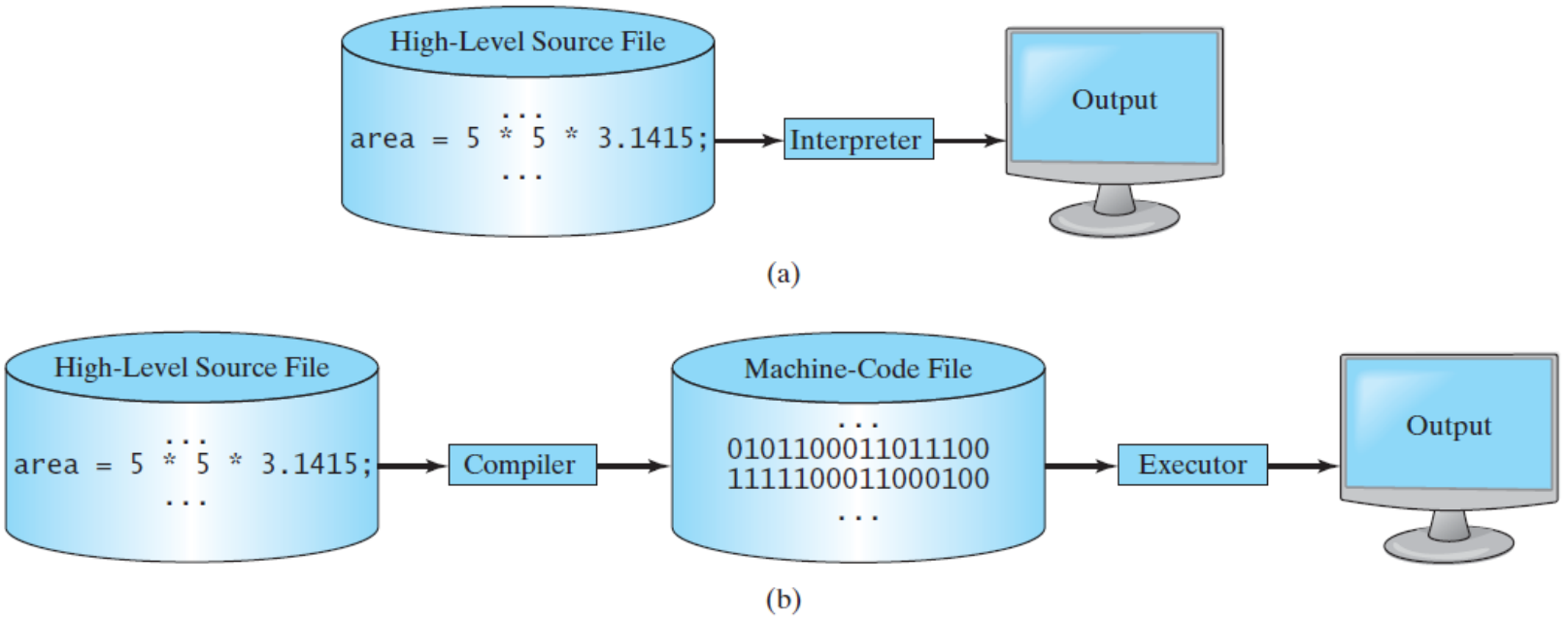
Python (1991)

- **Python is evolving …**
- **The goal of this course:**
  - ~~Keep track of recent updates~~ ✗
  - Provide you a comprehensive knowledge base ✓
  - Our students have very diverse backgrounds …

Python release cycle

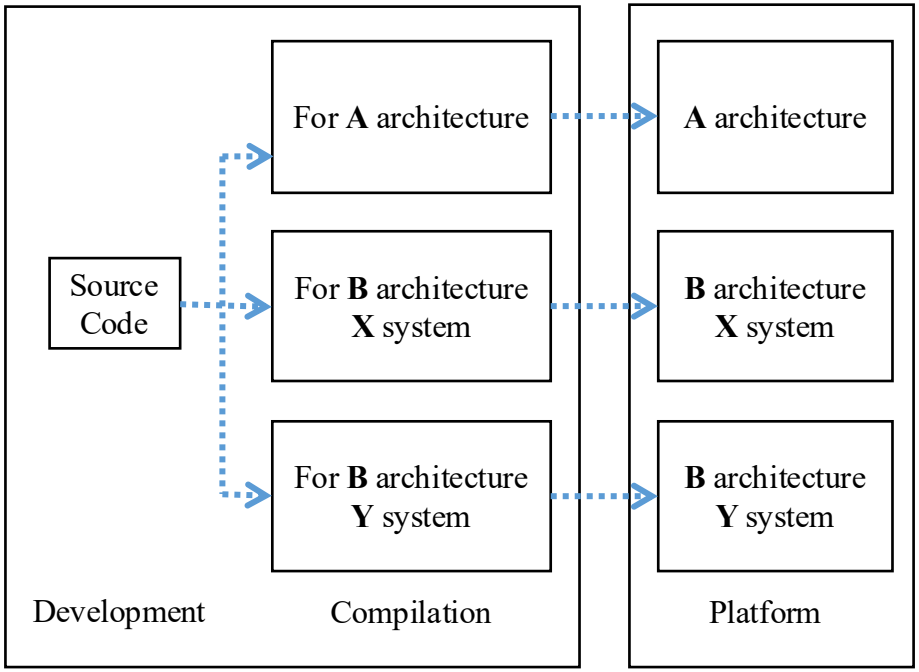


Interpreter v.s. compiler

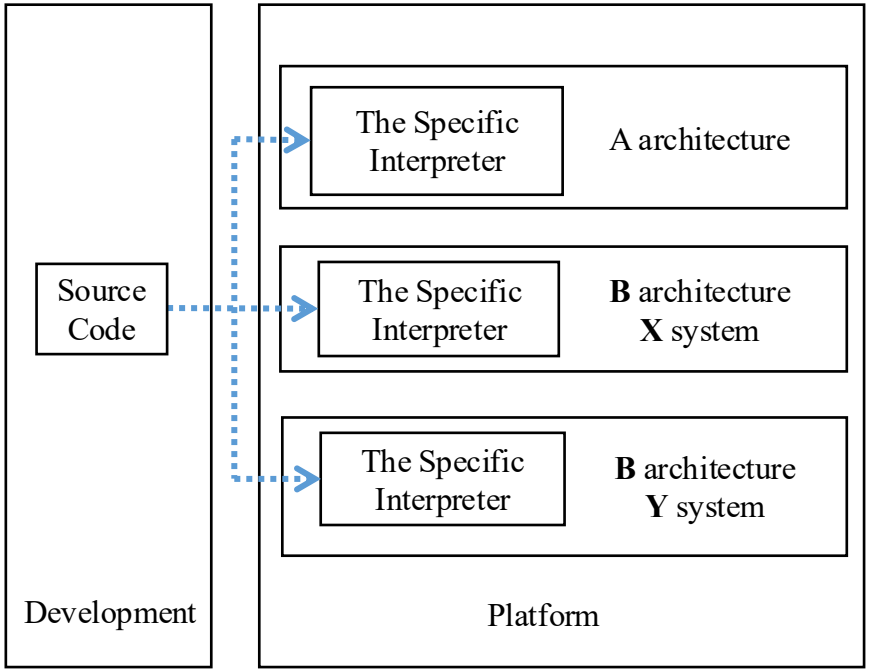


Interpreter v.s. compiler

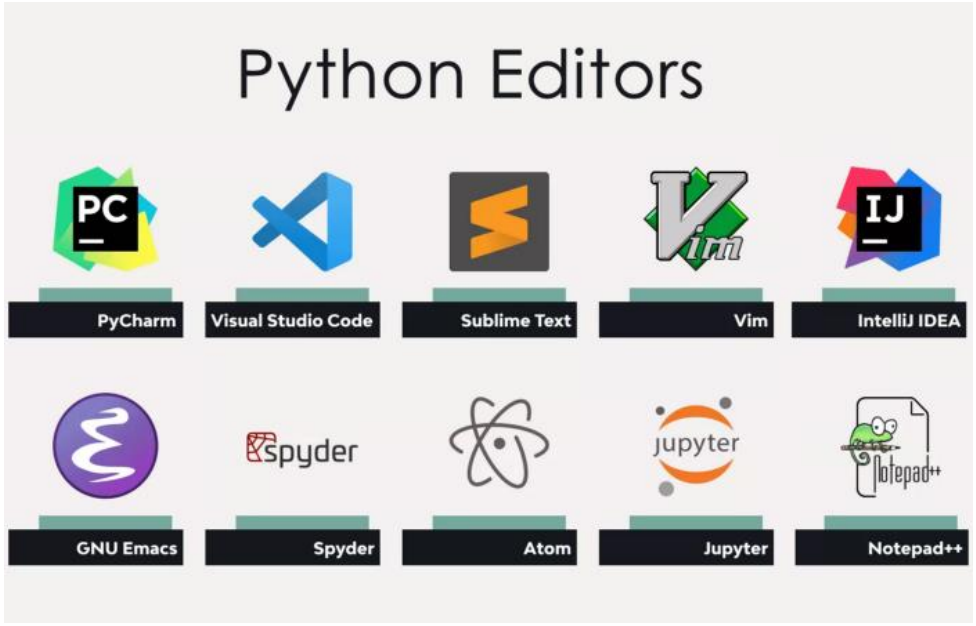
Compiler Based Languages
C, C++



Interpreter Based Languages
Python



Integrated development environment

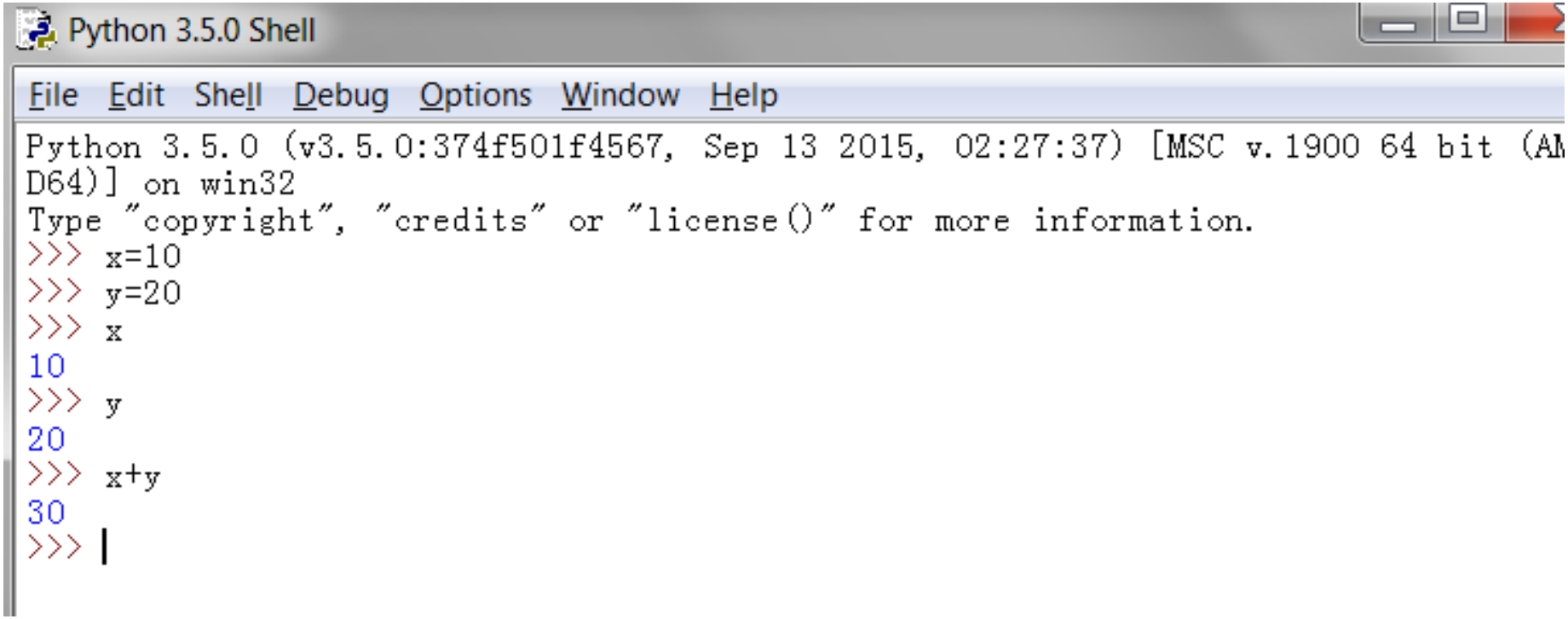


Moderne™

Elements of Python Language

- Vocabulary/words – Variables and Reserved words
- Sentence structure – valid syntax patterns
- Story structure – constructing a meaningful program for some purposes

Variables


Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=10
>>> y=20
>>> x
10
>>> y
20
>>> x+y
30
>>>

Variables

- A variable is a named space in the memory where a programmer can store data and later retrieve the data using the variable name
- The value of a variable can be changed later in a program
- Variables takes memory to store
- You cannot use the following words as variables

```
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
```

Constants

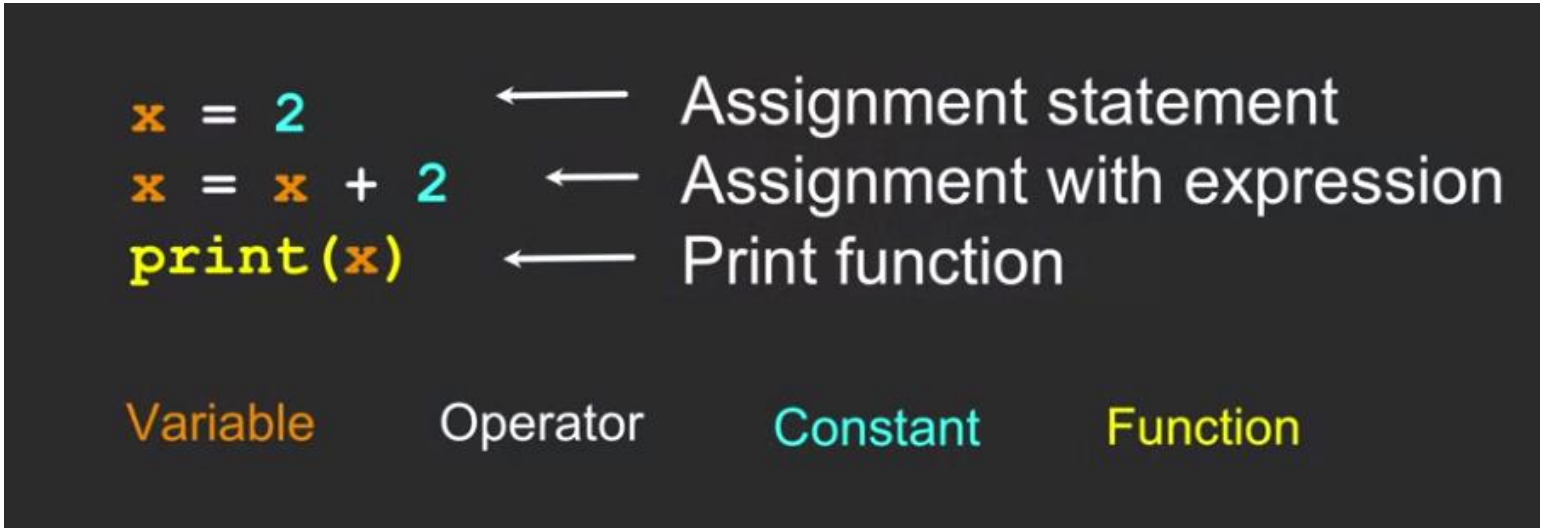
- Fixed values such as numbers and letters are called constants, since their values won't change
- String constants use single-quotes (') or double-quotes (")

Sentences or lines

Each line in python is an individual statement

Like a recipe, a sequence of steps to be done in pre-determined order

Steps can be conditional, repeated or stored to be used over and over again



Assignment

- **Assignment statement**

  Assigned values can be retrieved from located memory

$$x = 100 - 10 + x*3 - x/10$$

- **Cascaded assignment**

  Multiple variables can be set as the same value using single assignment statement

```
>>> z = y = x = 2 + 7 + 2
>>> x, y, z
(11, 11, 11)
```

- **Simultaneous assignment**

  Values of two variables can be exchanged

```
>>> c = "deepSecret"          # Set current password.
>>> o = "you'll never guess" # Set old password.
>>> c, o                      # See what passwords are.
('deepSecret', "you'll never guess")
>>> c, o = o, c               # Exchange the passwords.
```

Practice

- Write a program to exchange the values of two variables without using simultaneous assignment

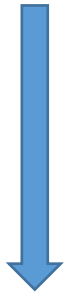
Numeric expression and operators

- Classic math operators

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

- Operator precedence rules

✓ Parenthesis are always with highest priority
✓ Power
✓ Multiplication, division and remainder
✓ Addition and subtraction
✓ Left to right



- More Operations

Floor division:	256 // 10	return	25
Divmod:	divmod(143, 25)	return	5, 18
Augmented Assignment:	x <op>= 7	equals	x = x <op> 7

Practice

- Describe the precedence of following statement

x = 1+2**3/4*5

Data Type

- Some operations are prohibited on certain types
- Check data type using function type()
- Type of a variable can be dynamically changed, and determined by last assignment

```
x = 5              # int
x = x + 3.14       # float
```

- Common data types
  - Integer: 1,2,100,11550
  - Float: 2.5, 7.8, 79.99
  - String: “cuhksz”, “python”
  - Boolean: True False

- Data type conversions
  - Integer will be converted into float implicitly when operating float and integer
  - Use int() & float() to convert other data type to integer & float
  - You’ll get an Error if source type is string and it contains character other than numbers

Useful functions

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

```
Who are you?
Chuck
Welcome Chuck
```

- Input()
  - Stop program flow and Wait for user input
  - Return a string
- Print()
  - Output to console or screen
- Eval()
  - Takes a string argument and evaluate it as a Python expression, returns the result
  - Be cautious about using it when users can cause problems with "inappropriate" input

- Comments
  - Anything after a "#" is ignored by python

  Why comment?

  ✓Describe what is going to happen in a sequence of code

  ✓Document who wrote the code and other important information

  ✓Turn off a line of code – usually temporarily

Examples

```
>>> string = "5 + 12" # Create a string.
>>> print(string)      # Print the string.
5 + 12
>>> eval(string)       # Evaluate the string.
17
>>> print(string, "=", eval(string))
5 + 12 = 17
>>> eval("print('Hello World!')") # Can call functions from eval().
Hello World!
>>> # Using eval() we can accept all kinds of input...
>>> age = eval(input("Enter your age: "))
Enter your age: 57.5
>>> age
57.5
>>> age = eval(input("Enter your age: "))
Enter your age: 57
>>> age
57
>>> age = eval(input("Enter your age: "))
Enter your age: 40 + 17 + 0.5
>>> age
57.5
```

Flow Control

Boolean type

- Python contains a built-in Boolean type, which takes two values True/False
- Number 0 can also be used to represent False. All other numbers represent True

Comparison operators

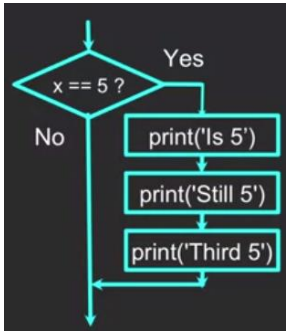
- Boolean expressions ask a question and produce a Yes/No result
- Comparison operators check variables while not change values
- Common comparison operators

Operator	Meaning
<	Smaller
<=	Smaller or equal
==	Equal
>=	Larger or equal
>	Larger
!=	Not equal

If-Else statement

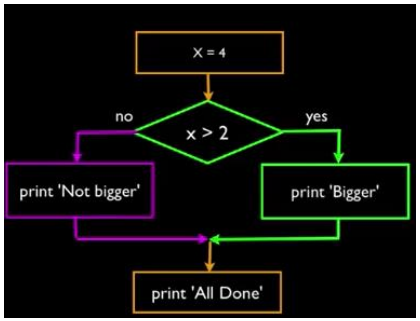
One way decisions

```
x=5
print('Before 5')
if x==5:
    print('Is 5')
    print('Is still 5')
    print('Third 5')

print('Afterwards 5')

print('Before 6')
if x==6:
    print('Is 6')
    print('Is still 6')
    print('Third 6')

print('Afterwards 6')
```



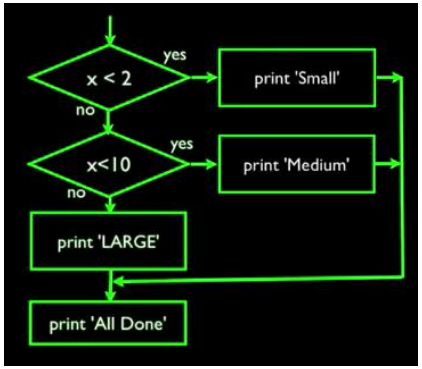
Two way decisions

```
x=1

if x>2:
    print('Bigger')
else:
    print('Smaller')

print('Finished')
```



Multi way decisions

```
x=2
if x<2:
    print('Small')
elif x<10:
    print('Medium')
else:
    print('Large')

print('Finished')
```



Tips on if - else

```
x=1

if x>2:
    print('Bigger')
else:
    print('Smaller')

print('Finished')
```

```
x=1

if x>2:
    print('Bigger')
    else:
    print('Smaller')

print('Finished')
```

- Else must come after if
- Use indentation to match if and else

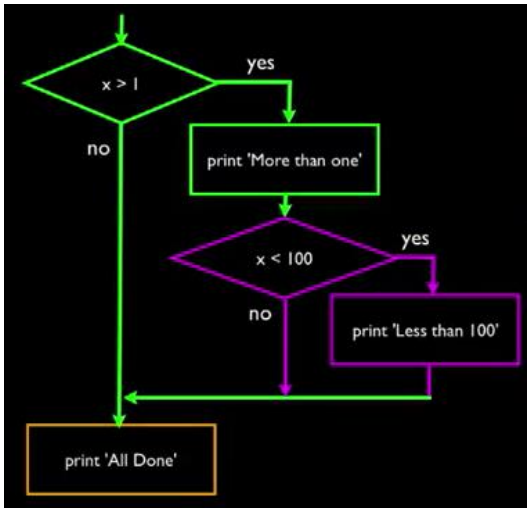
Indentation

- Increase after if/for statements
- Maintain to indicate the scope of the block
- Decrease to indicate the end of a block
- Blank lines and comments are ignored

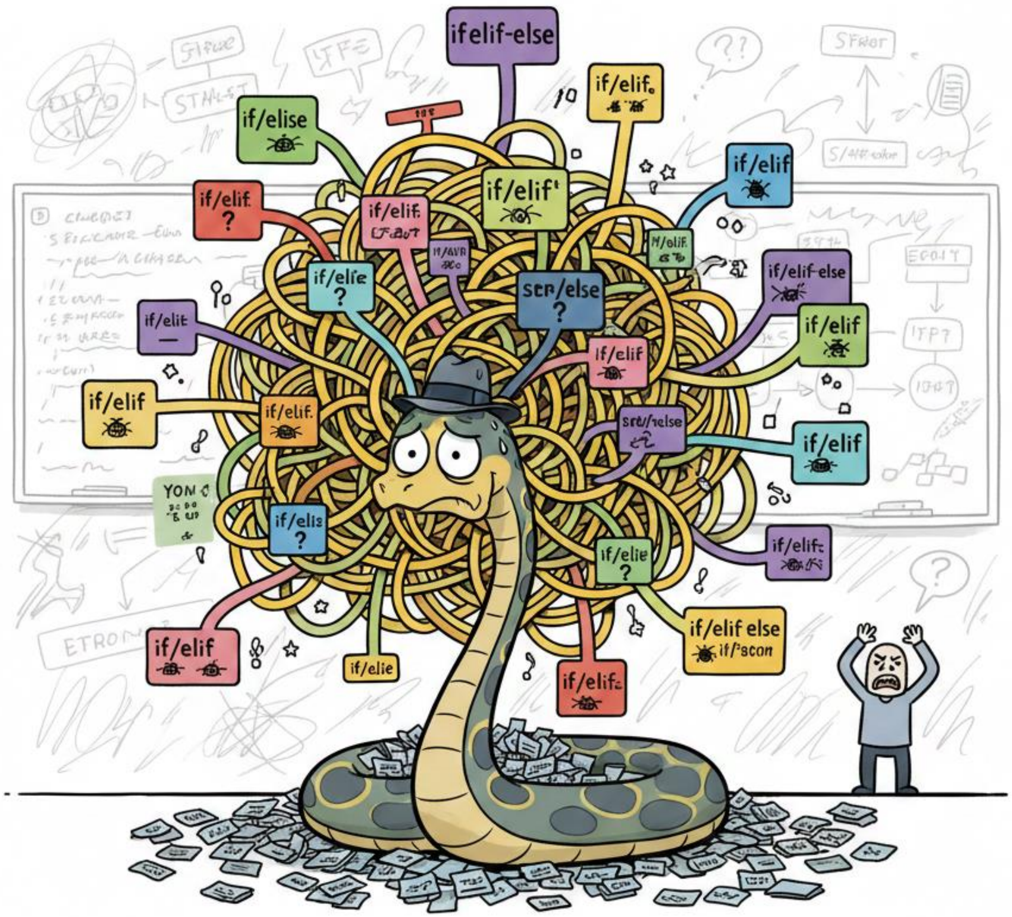
```
x=5
print('Before 5')
if x==5:
    print('Is 5')
    print('Is still 5')
    print('Third 5')

print('Afterwards 5')

print('Before 6')
if x==6:
    print('Is 6')
    print('Is still 6')
    print('Third 6')

print('Afterwards 6')
```

Nested decisions

```
x=42
if x>1:
    print('More than 1')

    if x<100:
        print('Less then 100')

print('Finished')
```



TOO MANY NESTED DECISIONS = DISASTER!



Logical operators

- Logical operators can be used to combine several logical expressions into a single expression

- Python has three logical operators: not, and, or

```
>>> not True
False
>>> False and True
False
>>> not False and True
True
>>> (not False) and True     # Same as previous statement.
True
>>> True or False
True

>>> not False or True        # Same as: (not False) or True.
True
>>> not (False or True)
False
>>> False and False or True  # Same as: (False and False) or True.
True
>>> False and (False or True)
False
```

Try/except structure

- You surround a dangerous part of code with try/except
- If the code in try block works, the except block is skipped
- If the code in try block fails, the except block will be executed

Use try/except to capture errors

```
astr = 'Hello bob'
try:
    istr = int(astr)
except:
    istr = -1
print('First', istr)

astr = '123'
try:
    istr = int(astr)
except:
    istr = -1
print('Second', istr)
```

- When the first conversion fails, it just stops into the except block, and the program continues
- When the second conversion succeeds, it just skips the except block

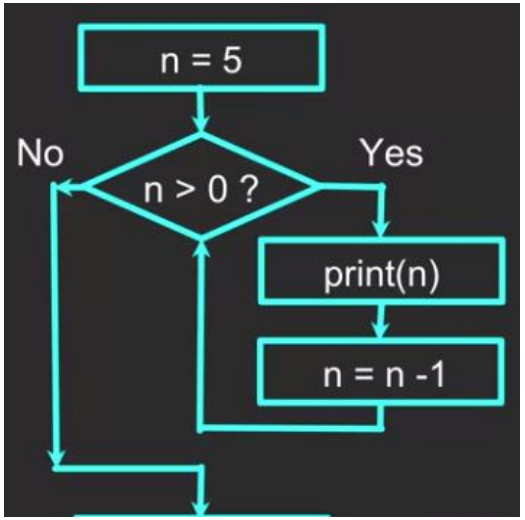
Repeated Flow - Loop

Program

```
n=5
while n>0:
    print(n)
    n = n - 1
print("Finish")
```

Outputs

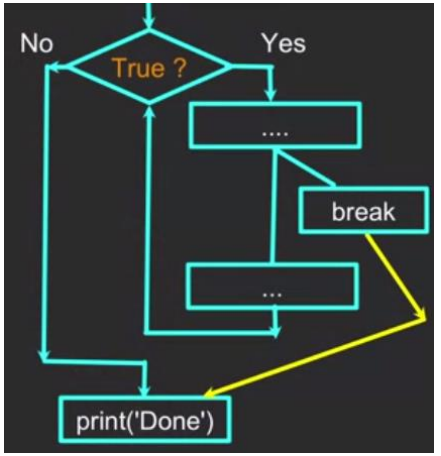
```
5
4
3
2
1
Finish
>>>
```



- Loops (repeated steps) have iterative variables that change each time through a loop
- Often these iterative variables go through a sequence of numbers

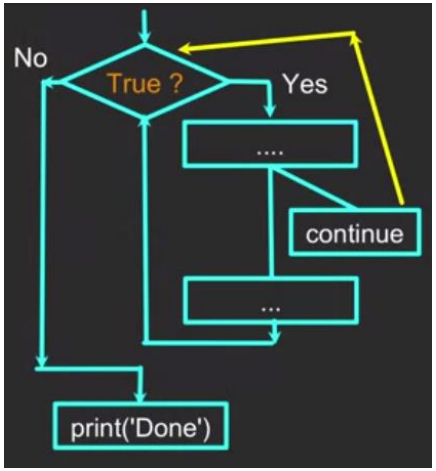
Repeated Flow - Loop

```
while (True):
    line = input('Enter a word:')
    if line == 'done':
        break
    print(line)
print('Finished')
```



Breaking from a Loop

```
while True:
    line = input('Input a word:')
    if line[0] == '#': continue
    if line == 'done':
        break
    print(line)
print('Done')
```



Finishing an iteration with continue

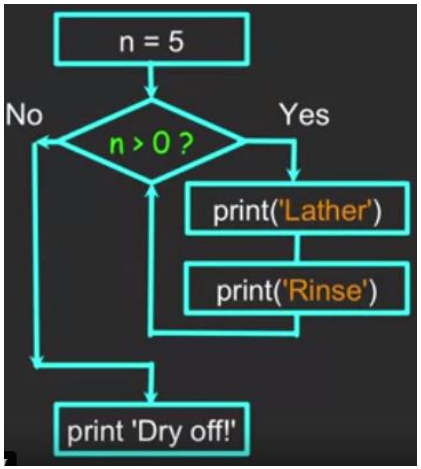
Indefinite Loop

- While loops are called "indefinite loops", since they keep going until a logical condition becomes false

- Till now, the loops we have seen are relatively easy to check whether they will terminate

- Sometimes it can be hard to determine whether a loop will terminate

```
n=5
while n>0:
    print('Lather')
    print('Rinse')
n=n-1
print('Dry off!')
```



Definite loop

- Quite often we have a finite set of items
- We can use a loop, each iteration of which will be executed for each item in the set, using the for statement
- These loops are called "definite loops" because they execute an exact number of times
- It is said that "definite loops iterate through the members of a set"
- For loops (definite loops) have explicit iteration variables that change each time through a loop.

Example

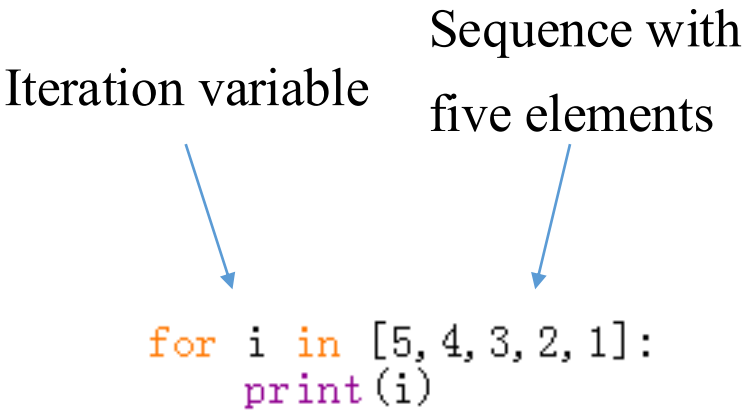
```
for i in [5, 4, 3, 2, 1]:
    print(i)
print('Finished')
```

Output

```
5
4
3
2
1
Finished
```

in

- The iteration variable "iterates" through a sequence (ordered set)
- The block (body) of the code is executed once for each value in the sequence
- The iteration variable moves through all of the values in the sequence



Practice

- **Given a list of numbers, write a program to calculate their sum using for loop**

Practice

- **Given a list of numbers, write a program to calculate their sum using for loop**

```
numberSet = [3, 4, 98, 38, 9, 10, 199, 78]

total = 0
print('Before', total)
for num in numberSet:
    total = total + num
    print(total, num)
print('Last', total)
```

```
Before 0
3 3
7 4
105 98
143 38
152 9
162 10
361 199
439 78
Last 439
```

Filtering in a loop

- We can use an if statement in a loop to catch/filter the values we are interested in
- Operations shall be considered whenever filtering conditions satisfied or not

```
smallest_so_far = None
print('Before',smallest_so_far)

for num in [9,39,21,98,4,5,100,65]:
    if smallest_so_far == None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far,num)

print('After',smallest_so_far)
```

```
Before None
9 9
9 39
9 21
9 98
4 4
4 5
4 100
4 65
After 4
```

Finding the smallest number

Function

Python functions

- There are two types of functions in Python
  - ✓ Built-in functions which are part of Python, such as print(), int(), float(), etc
  - ✓ Functions that we define ourselves and then use

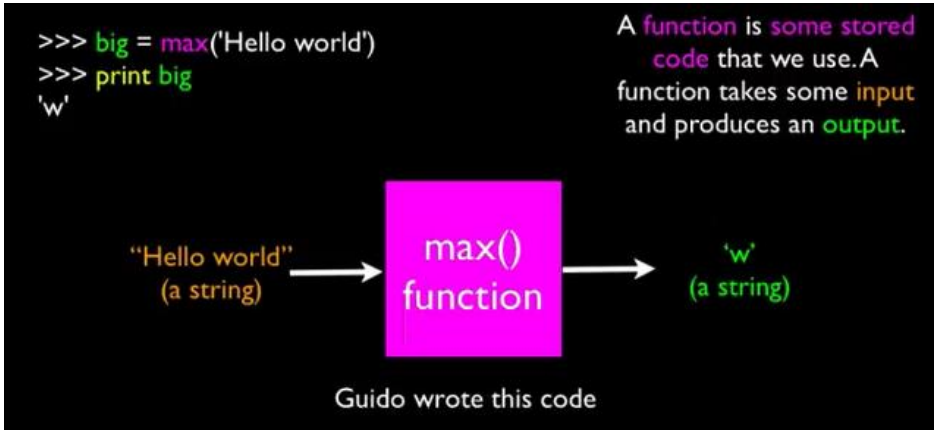
- The names of built-in functions are usually considered as new reserved words, i.e. we do not use them as variable names

Function definition

- In Python, a function is some reusable code which can take arguments as input, perform some computations, and then output some results

- Functions are defined using reserved word def

- We call/invoke a function by using the function name, parenthesis and arguments in an expression



Example - max()

Building our own functions

- We create a new function using the def key word, followed by optional parameters in parenthesis
- We indent the body of the function
- This defines the function, but does not execute the body of the function

```
x=5
print('Hello')

def print_lyrics():
    print('I am a lumberjack, and I am okay.')
    print('I sleep all night and I work all day.')

print('Yo')
print_lyrics()
x=x+2
print(x)
```

```
Hello
Yo
I am a lumberjack, and I am okay.
I sleep all night and I work all day.
7
```

Argument

- An argument is a value we pass into the function as its input when we call the function
- We use arguments so we can direct the function to do different kinds of work when we call it at different times
- We put the argument in parenthesis after the name of the function

big = max('I am the one')

Parameters

- A parameter is a variable which we use in the function definition that is a 'handle' that allows the code in the function to access the arguments for a particular function invocation

```
def greet(lang):
    if lang=='es':
        print('Hola')
    elif lang=='fr':
        print('Bonjour')
    else:
        print('Hello')
```

Multiple parameters/arguments

- We can define more than one parameter in a function definition
- We simply add more arguments when we call the function
- We match the number and order of arguments and parameters

```
def AddTwo(a, b):
    total = a+b
    return total

x=AddTwo(3, 5)
print(x)
```

Default argument

- Arguments can be set as default argument values
- Default argument values will work when function is invoked without respective arguments

```
def printArea(width = 1, height = 2):
    area = width * height
    print("width:", width, "\theight:", height, "\tarea:", area)

printArea() # Default arguments width = 1 and height = 2
printArea(4, 2.5) # Positional arguments width = 4 and height = 2.5
printArea(height = 5, width = 3) # Keyword arguments width
printArea(width = 1.2) # Default height = 2
printArea(height = 6.2) # Default width = 1
```

Return values

- Often a function will take its arguments, do some computation and return a value to be used as the value of the function call in the calling expression. The return keyword is for this purpose.
- The return statement ends the function and send back the result of the function

```
def greet(lang):
    if lang=='es':
        return 'Hola'
    elif lang=='fr':
        return 'Bonjour'
    else:
        return 'Hello'
```

```
>>> print(greet('en'),'Glenn')
Hello Glenn
>>> print(greet('es'),'Sally')
Hola Sally
>>> print(greet('fr'),'Michael')
Bonjour Michael
```

Return values

Void functions

- Void function is a function that does not return any value
- When a function has no return statement, it will return None

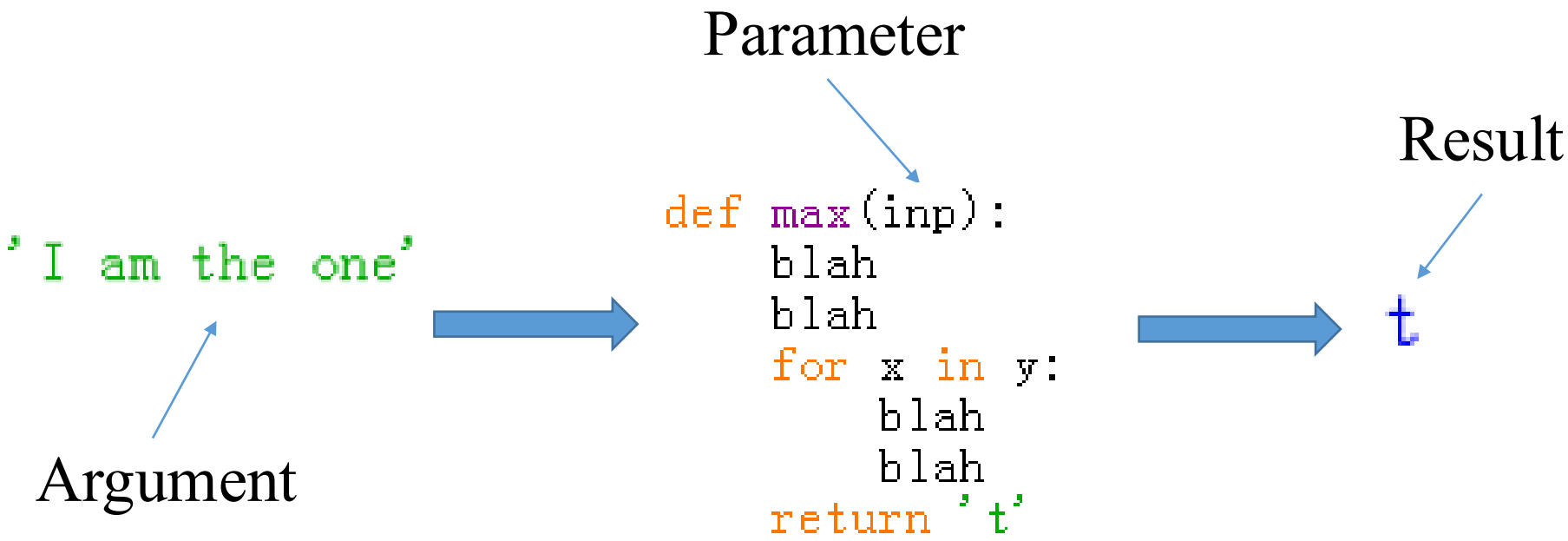
Return multiple values

- Python allows a function to return multiple values
- you need to pass the returned values in a simultaneous assignment

```
def sort(number1, number2):
    if number1 < number2:
        return number1, number2
    else:
        return number2, number1

n1, n2 = sort(3, 2)
print("n1 is", n1)
print("n2 is", n2)
```

Argument, parameter, and result

```
>>> big = max('I am the one')
>>> print(big)
t
```



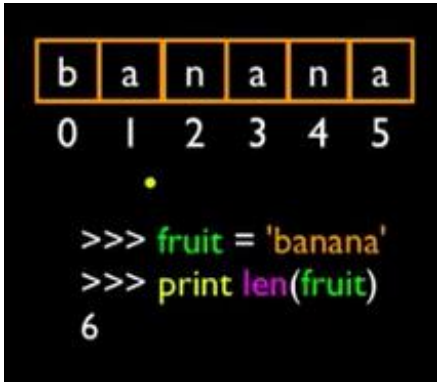
To function or not function…

- Organize your code into paragraphs - capture a complete thought and name it
- Don't repeat yourself – name it to work once and reuse it
- If something goes too complex, break up them into several blocks, and put each of them into a function
- Make a library of common stuffs that you do over and over again – perhaps share with other people

String type

- A string is a sequence of characters
- A string literal uses quotes ‘’ or “”
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string

- String can be indexed to get any character within it
- Index number must be an integer which starts from zero
- String has a length, which can be get by built-in function len()



- Python error if you attempt to index beyond the end of a string

Slicing strings

- We can look at ant continuous section of a string using colon operator
- The second number is one beyond the end of the slice, which is not included
- If the second number is greater than the length of the string, it stops at the end
- If the first or second number of the slice is not specified, it is assumed to be the beginning or end of the string, respectively


```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```

```
>>> s='Monty Python'
>>> print(s[:6])
Monty
>>> print(s[3:])
ty Python
>>> print(s[:])
Monty Python
```

String library

- Python has a number of string functions which are in the string library
- These functions are built-into every string, we invoke them by appending the function to the string variable
- These function do not modify the original string, instead they return a new string altered from the original string

```
>>> stuff = 'hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '_
_eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs
__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__'
, '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__r
epr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subcl
asshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isd
ecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'issp
ace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'pa
rtition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip
', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate
', 'upper', 'zfill']
```

https://docs.python.org/3/library/stdtypes.html#string-methods

Searching a string

- Use find() function to search for a substring in a string
- find() finds the first occurrence of target substring
- If not found, find() returns -1

```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

Making a string upper/lower

- A string can be convert into all upper case or lower case using upper() or lower()

```
>>> myStr = 'I am the one, I will beat Matrix'
>>> newStr = myStr.upper()
>>> print(newStr)
I AM THE ONE, I WILL BEAT MATRIX
>>> newStr = myStr.lower()
>>> print(newStr)
i am the one, i will beat matrix
```

Search and Replace

- The replace() function is like a "search and replace" operation in a word processor
- It replaces all occurrences of the search string with the replacement string

```
>>> greet = 'Hello, Bob'
>>> newStr = greet.replace('Bob', 'Jane')
>>> print(newStr)
Hello, Jane
>>> newStr = greet.replace('o','X')
>>> print(newStr)
HellX, BXb
>>> newStr = greet.replace('z','X')
>>> newStr
'Hello, Bob'
```

Stripping whitespaces

- Use strip method to remove whitespaces at the beginning and/or end
- lstrip() and rstrip() processes the left and right end of a string respectively
- strip() removes whitespaces at both ends

```
>>> greet = '   Hello Bob   '
>>> greet.lstrip()
'Hello Bob   '
>>> greet.rstrip()
'   Hello Bob'
>>> greet.strip()
'Hello Bob'
```

Using 'in' in conditional statement

- The in keyword can also be used to check whether one string is in another string
- The in expression is a logical expression and returns True or False
- It can be used in if or while statement

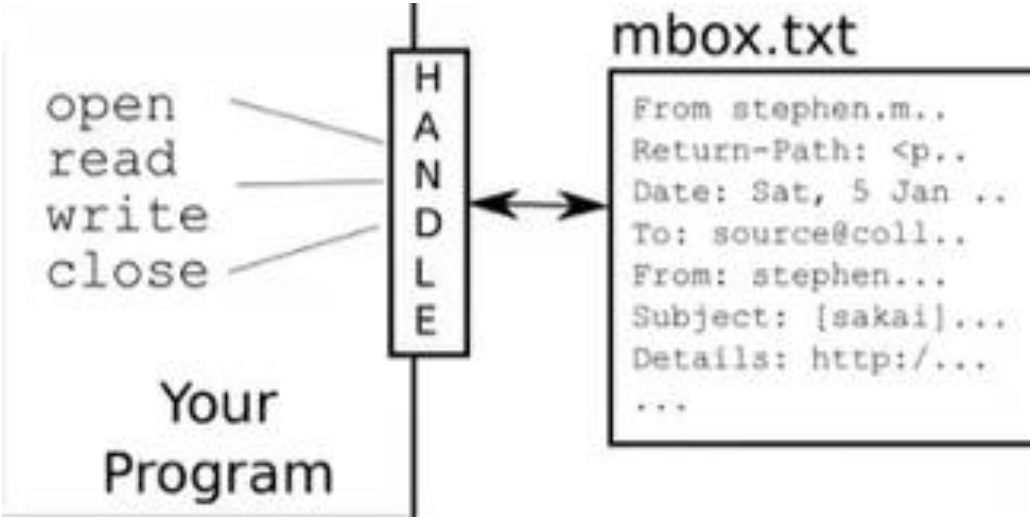
```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit:
        print('Got cha!')

Got cha!
```

File processing

- open() : Before we can read the contents of a file, we must tell Python which file we are going to work with and what we will do with that file
- open() returns a "file handle" - a variable used to perform operations on files
- handle = open(filename, mode)
- Filename is a string; Mode is optional, use 'r' if we want to read the file, and 'w' if we want to write to the file
- Error if the input filename not exists

Handle

```
>>> fhand = open('c:\Python35\myhost.txt','r')
>>> print(fhand)
<_io.TextIOWrapper name='c:\\Python35\\myhost.txt' mode='r' encoding='cp936'>
```



File processing

- A text file can be thought of as a sequence of lines
- A text file has newline at the end of each line

- We use a new character to indicate when a line ends called "newline"
- We represent it as '\n' in strings; Newline is still one character, not two

```
>>> stuff = 'Hello\nWorld'
>>> stuff
'Hello\nWorld'
>>> print(stuff)
Hello
World
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

File Reading

File handle as a sequence

- A file handle open for read can be treated as a sequence of strings where each line in the file is a string in the sequence
- We can use the for statement to loop through a sequence

```
fhand = open('myhost.txt','r')

for line in fhand:
    print(line)

fhand.close()
```

Read the whole file

- Read the whole file into a single string

```
fhand = open('myhost.txt','r')
allText = fhand.read()
print('The length of the file:',len(allText))
print('The first 20 characters of the file:', allText[:20])
```

File Writing

- To write a file, use the open() function with 'w' argument
- Use the write() method to write to the file

```
fhand = open('test.txt','w')
fhand.write('The first line\n')
fhand.write('The second line\n')
fhand.write('The third line\n')
fhand.close()
```

Practice

A file named "score.txt" is shown below which contains a sequence of lines. Each line starts from the name of a student and then shows the scores of 5 courses one by one (note that the order of courses is random). Please write a program to 1) calculate the average score for each course; 2) write them to "average.txt" (for each line, write the course name followed by the average score).

score.txt

```
1  Jerry CSC1001:97 CSC1002:78 CSC1003:89 CSC1004:66 CSC1005:77
2  Michale CSC1002:88 CSC1001:95 CSC1003:77 CSC1005:87 CSC1004:66
3  Michelle CSC1003:54 CSC1001:99 CSC1004:92 CSC1005:90 CSC1002:95
4  Mike CSC1001:57 CSC1003:78 CSC1004:67 CSC1005:88 CSC1002:76
5
6
```

List, Dictionary & Tuple

List is a kind of Collection

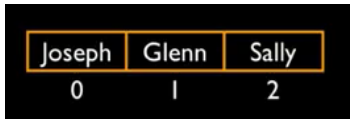
- A collection allows us to put many values in a single "variable"
- A collection is nice because we can carry all many variables around in one convenient package
- Not a collection: variables have only one value in them – when we put a new value in the variable, the old value will be over-written

```
>>> x=2
>>> x=4
>>> print(x)
4
```

List constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object – even another list
- A list can be empty
- Like string, list can be indexed
- Unlike string, list is mutable, whose element can be changed by indexing
- Length of list can be obtained using len()

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red','yellow','blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print(1, [5, 6], 7)
1 [5, 6] 7
>>> print([])
[]
```




```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(friends[1])
Glenn
```

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    fruit[0] = 'b'
TypeError: 'str' object does not support item assignment
>>>
>>> x=fruit.lower()
>>> print(x)
banana
>>>
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2]=28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

List methods

Build a list from scratch

- We can create an empty list using list(), then add elements using append()
- List stays in order, and new elements are positioned at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

Concatenating lists

- Addition between two lists creates a new list, who is concatenation of original ones

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> c=a+b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

List methods

Slicing list

- List can be sliced using colon; the first number is start position while the second is "up to but not included"

```
>>> t=[9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41, 12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Sort list

- A list is an ordered sequence, and hold items in order until order is changes
- A list can be sorted using sort(), meaning "sort yourself"

```
>>> friend = ['Tom','Jerry','Bat']
>>> friends.sort()
>>> print(friends)
['Bat', 'Jerry', 'Tom']
>>> print(friends[1])
Jerry
>>>
>>> numbers = [1, 2, 5, 100, 32, 7, 97, 1001]
>>> numbers.sort()
>>> print(numbers)
[1, 2, 5, 7, 32, 97, 100, 1001]
```

List methods

Build-in functions of list

```
>>> x=list()
>>> type(x)
<class 'list'>
>>> dir(x)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__'
, '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
, '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__'
, '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_e
x__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__s
izeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'ex
tend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

Practice

- Write a program to instruct the user to input several numbers and calculate their average using list methods

```
numlist = list()
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(numlist)
print('Average:', average)
```

Best friends: strings and lists

- Use the split() method to break up a string into a list of strings
- We think of these as words
- We can access a particular word or loop through all the words

```
>>> myStr = 'Catch me if you can'
>>> words = myStr.split()
>>> print(words)
['Catch', 'me', 'if', 'you', 'can']
>>> print(len(words))
5
>>> print(words[0])
Catch
>>> for w in words: print(w)

Catch
me
if
you
can
```

- When you do not specify a delimiter, multiple spaces are treated like "one" delimiter
- You can specify what delimiter character to use in splitting

```
>>> line = 'A lot                of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> len(thing)
1
>>>
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
```

- List: a linear collection of values that stay in order
- Dictionary: a "bag" of values, each with its own label

Dictionary

- Dictionaries are Python's most powerful data collection
- Dictionaries allow us to do fast database-like operations in Python
- Dictionaries have different names in different languages
  - Associative arrays – Perl/PHP
  - Properties or Map or HashMap – Java
  - Property Bag – C#/.Net

- Dictionary literals use curly braces and have list of key:value pairs
- You can make an empty dictionary using empty curly braces or dict()

```
>>> jjj = {'chuck':1,'fred':42, 'jan':100}
>>> print(jjj)
{'fred': 42, 'chuck': 1, 'jan': 100}
>>> ooo={}
>>> print(ooo)
{}
```

List v.s. dictionary

- Lists are ordered sequences while dictionaries have no order
- Lists index based on positions while dictionaries index based on key

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age']=21
>>> ddd['course']=182
>>> print(ddd)
{'age': 21, 'course': 182}
>>> ddd['age']=23
>>> print(ddd)
{'age': 23, 'course': 182}
```

- List gets error if index number beyond its length while diction gets error to reference a key which is not within it
- in operator can be used to see if an element is in list or a key is in the dictionary

The get() method

- This pattern of checking to see if a key is already in a dictionary, and assuming a default value if the key is not there is so common, that there is a method called get() that does this for us

```
>>> counts = {'aaa':1,'bbb':2,'ccc':5}
>>> print(counts.get('eee',0))
0
```

Definite loops and dictionaries

- Even though dictionaries are not stored in order, we can write a for loop that goes through all elements in a dictionary – actually it goes through all the keys in that dictionary and looks up the values

```
counts = {'chuck':1,'fred':42,'jan':100}

for key in counts:
    print(key,counts[key])
```

```
jan 100
fred 42
chuck 1
```

Definite loops and dictionaries

- You can get a list of keys, values or items (both) from a dictionary

```
>>> jjj = {'chuck':1,'fred':42,'jan':100}
>>> print(list(jjj))
['jan', 'fred', 'chuck']

>>> print(list(jjj.keys()))
['jan', 'fred', 'chuck']

>>> print(list(jjj.values()))
[100, 42, 1]
>>> print(list(jjj.items()))
[('jan', 100), ('fred', 42), ('chuck', 1)]
```

Definite loops and dictionaries

- We loop through the key-value pairs in a dictionary using two iteration variables
- Each iteration, the first variable is the key, and the second variable is the corresponding value for the key

```
counts = {'chuck':1,'fred':42,'jan':100}
for key,value in counts.items():
    print(key,value)
```

```
chuck 1
fred 42
jan 100
```

Tuples

- Tuples are another type of sequence that function more like a list – they have elements which are indexed starting from 0

```
>>> x=('Glenn','Sally','Joseph')
>>> print(x)
('Glenn', 'Sally', 'Joseph')
>>> y=(1,9,2)
>>> print(y)
(1, 9, 2)
>>> print(max(y))
9
```

```
>>> for i in y:
        print(i)

1
9
2
```

- Immutable: Unlike a list, once you create a tuple, you cannot change its contents – similar to a string
- Efficient: Since Python doe not have to build tuple structures to be modifiable, tuples are simpler and more efficient in terms of memory use and performance than lists

Tuple vs Dictionary

- The item() method in dictionaries returns a list of (key, value) tuples

```
>>> d=dict()
>>> d['csev']=2
>>> d['cwen']=4
>>> for (k,v) in d.items():
        print(k,v)


csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
>>> print(list(tups))
[('csev', 2), ('cwen', 4)]

>>> tups = list(tups)
>>> tups[1]
('cwen', 4)
```

Tuples are comparable

- The comparison operators work with tuples and other sequences if the first item is equal. Python goes on to the next element, until it finds the elements which are different

```
>>> (0, 1, 2)<(5, 1, 2)
True
>>> (0, 1, 200000)<(0, 3, 4)
True
>>> ('Jones','Sally')<('Jones','Fred')
False
>>> ('Jones','Sally')>('Adams','Sam')
True
```

Sorting dictionary via sorting tuple

- sort the dictionary by the key using the items() method

```
>>> d={'a':10,'b':1,'c':22}
>>> t=d.items()
>>> t=list(t)
>>> t
[('c', 22), ('b', 1), ('a', 10)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

- We can do this even more efficiently using a built-in function sorted() which takes a sequence as a parameter and returns a sorted sequence

```
>>> d={'a':10,'b':1,'c':22}
>>> d.items()
dict_items([('c', 22), ('b', 1), ('a', 10)])
>>> t=sorted(list(d.items()))
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

Practice

- Write a program, which sorts the elements of a dictionary by the value of each element

Sort by values instead of key

- If we could construct a list of tuples of the form (key, value) we could sort by value
- We do this with a for loop that creates a list of tuples

```
>>> d={'a':10,'b':1,'c':22}
>>> tmp = list()
>>> for k,v in d.items():
        tmp.append((v,k))

>>> print(tmp)
[(22, 'c'), (1, 'b'), (10, 'a')]
>>> tmp.sort(reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Thanks for Listening !