# CSC1002-Computational Laboratory

Kinley Lam

# CSC 1002 Week 1

Programming background

editor & tools

hello world

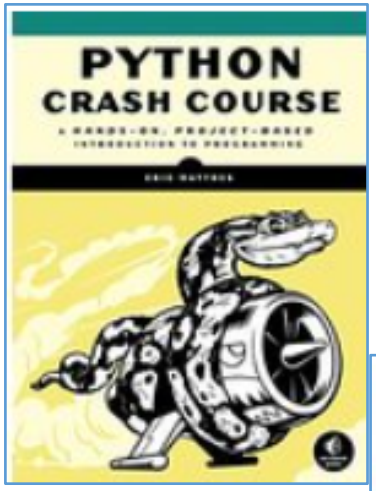
past assignments at a glance

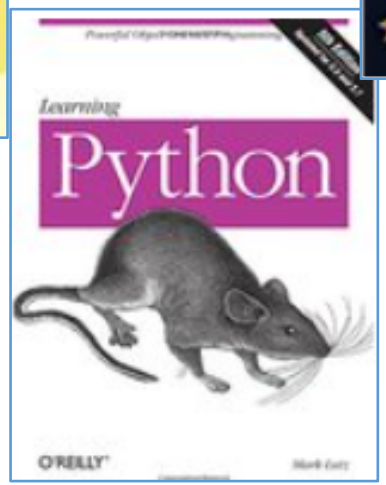
assessment

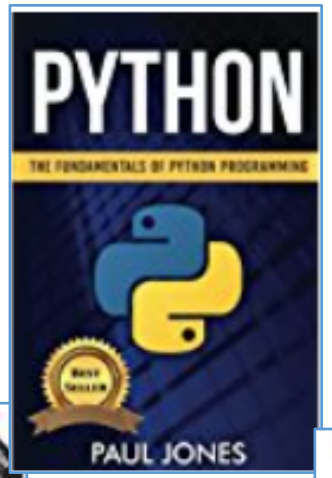
# No tutorials this week !!!

# References (books)

# References


PYTHON
CRASH COURSE

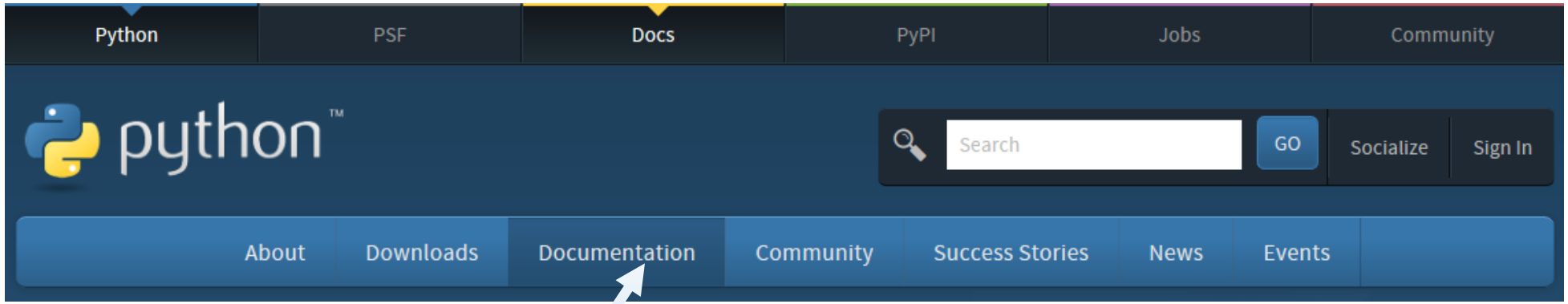

Learning
Python
O'REILLY


PYTHON
THE FUNDAMENTALS OF PYTHON PROGRAMMING
PAUL JONES


O'REILLY
Fluent
Python
CLEAR, CONCISE, AND EFFECTIVE PROGRAMMING
Luciano Ramalho


Python
Cookbook
O'REILLY

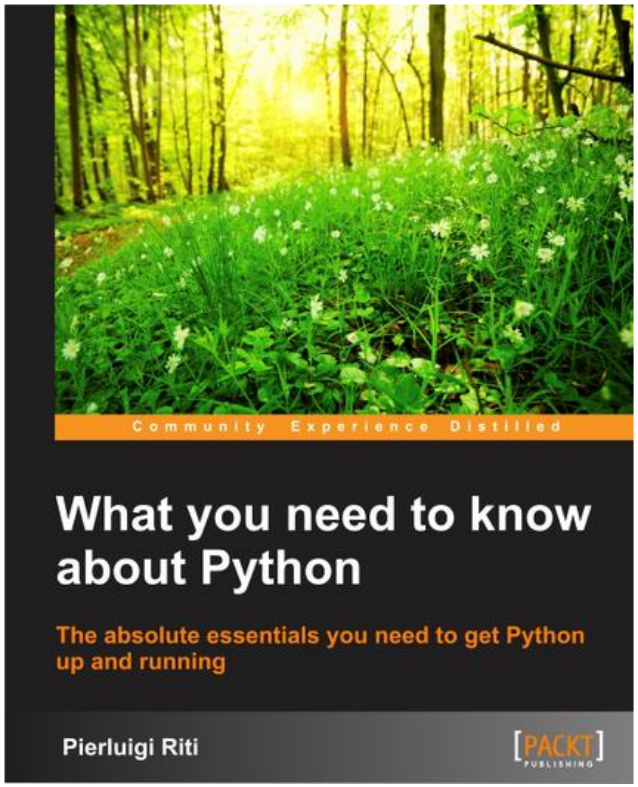
# python https://www.python.org/

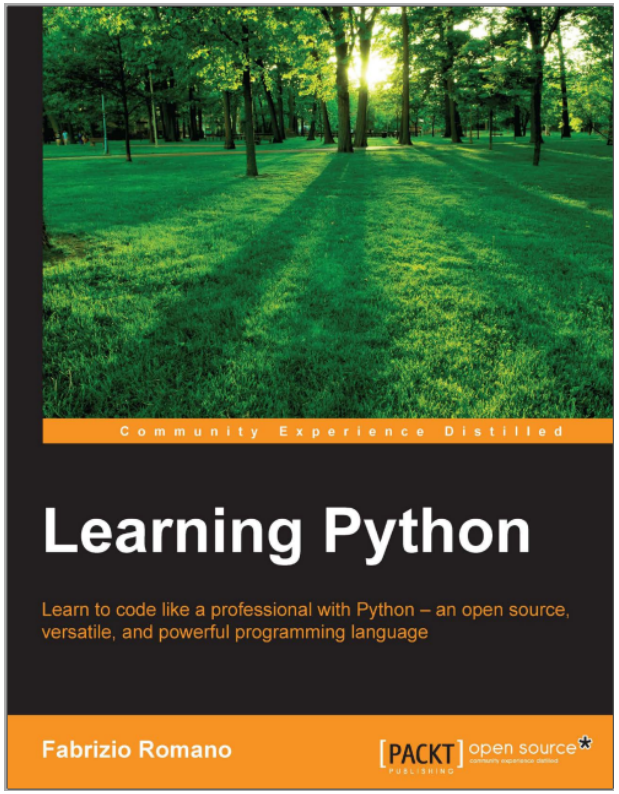



Follow “Python Books” and then “IntroductoryBooks” for a list of reference books and free e-book(s)

search for “Free eBook”

# Free ebooks via Python official website



# Programming Language

## Python 3.11.1 documentation

Welcome! This is the official documentation for Python 3.11.1.

**Parts of the documentation:**

### What's new in Python 3.11?
*or all "What's new" documents since 2.0*

### Tutorial
*start here*

### Library Reference
*keep this under your pillow*

### Language Reference
*describes syntax and language elements*

### Python Setup and Usage
*how to use Python on different platforms*

### Python HOWTOs
*in-depth documents on specific topics*

### Installing Python Modules
*installing from the Python Package Index & other sources*

### Distributing Python Modules
*publishing modules for installation by others*

### Extending and Embedding
*tutorial for C/C++ programmers*

### Python/C API
*reference for C/C++ programmers*

### FAQs
*frequently asked questions (with answers!)*

# hello world
# using notepad & word

# hello world

Write a “hello world” program


```
% python
>>> print('Hello world!')
Hello world!
```

# Why “Hello World”

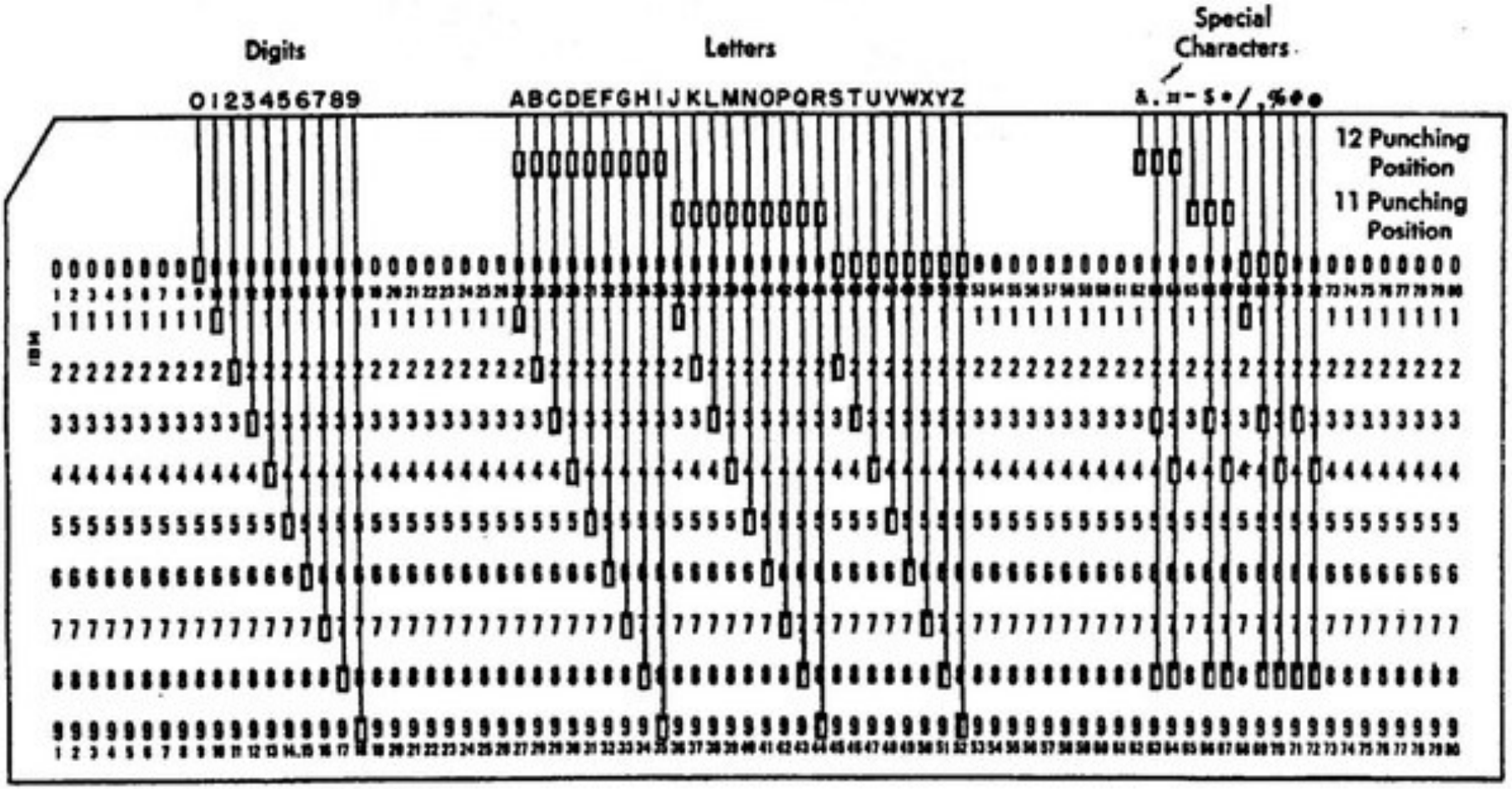
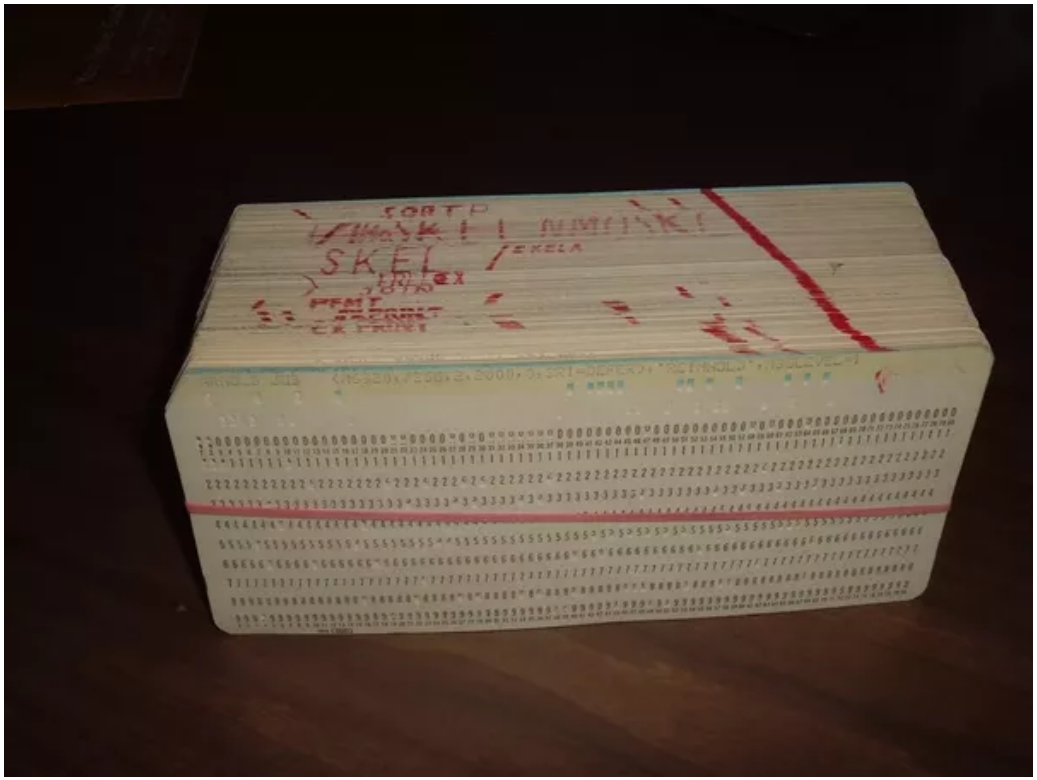
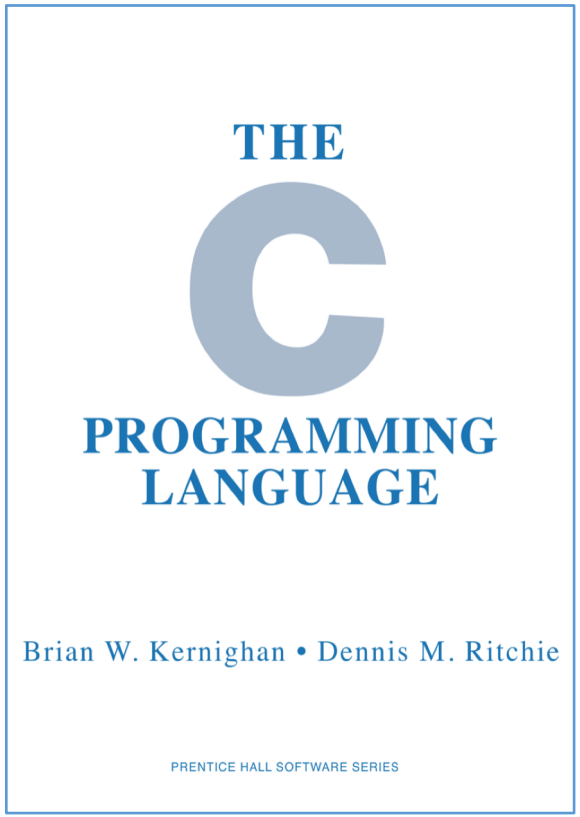


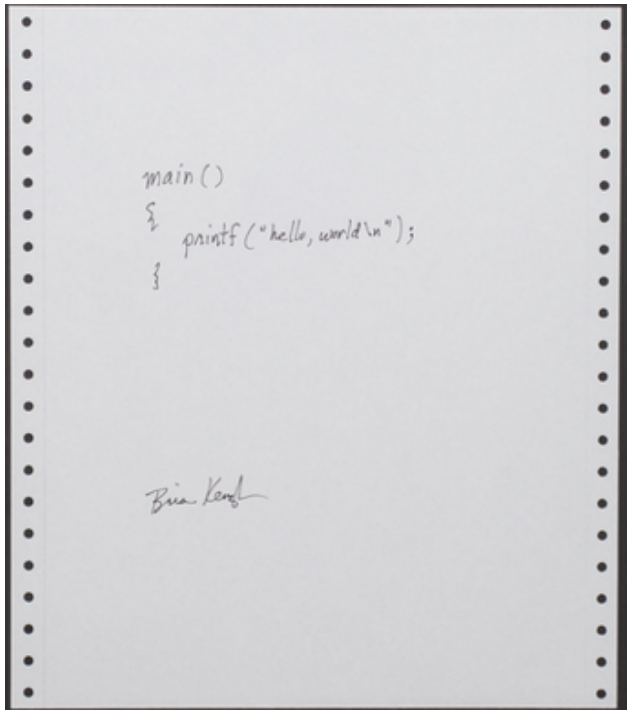

Figure 2. Punching Positions in Card




HYDROSTATICS
(METRIC)
INPUT
CREDIBILITY
BONJEANS
STATICAL
STABILITY

# Brian Kernighan




```
#include <stdio.h>

main( )
{
        printf("hello, world\n");
}
```

# Editor & tools

# IDE (Integrated Development Environment)

- Uniform interface – editing, execution and shell
- IntelliSense – auto code completion, quick info, parameter info … etc
- Code navigation – file explorer, code hierarchy, references, … etc
- Debugging, Source control, Refactoring, Unit testing … etc
- Extensions (tools, language support, interfaces .. Etc
- Many many others

# Source Code Editor





# Why Visual Studio Code

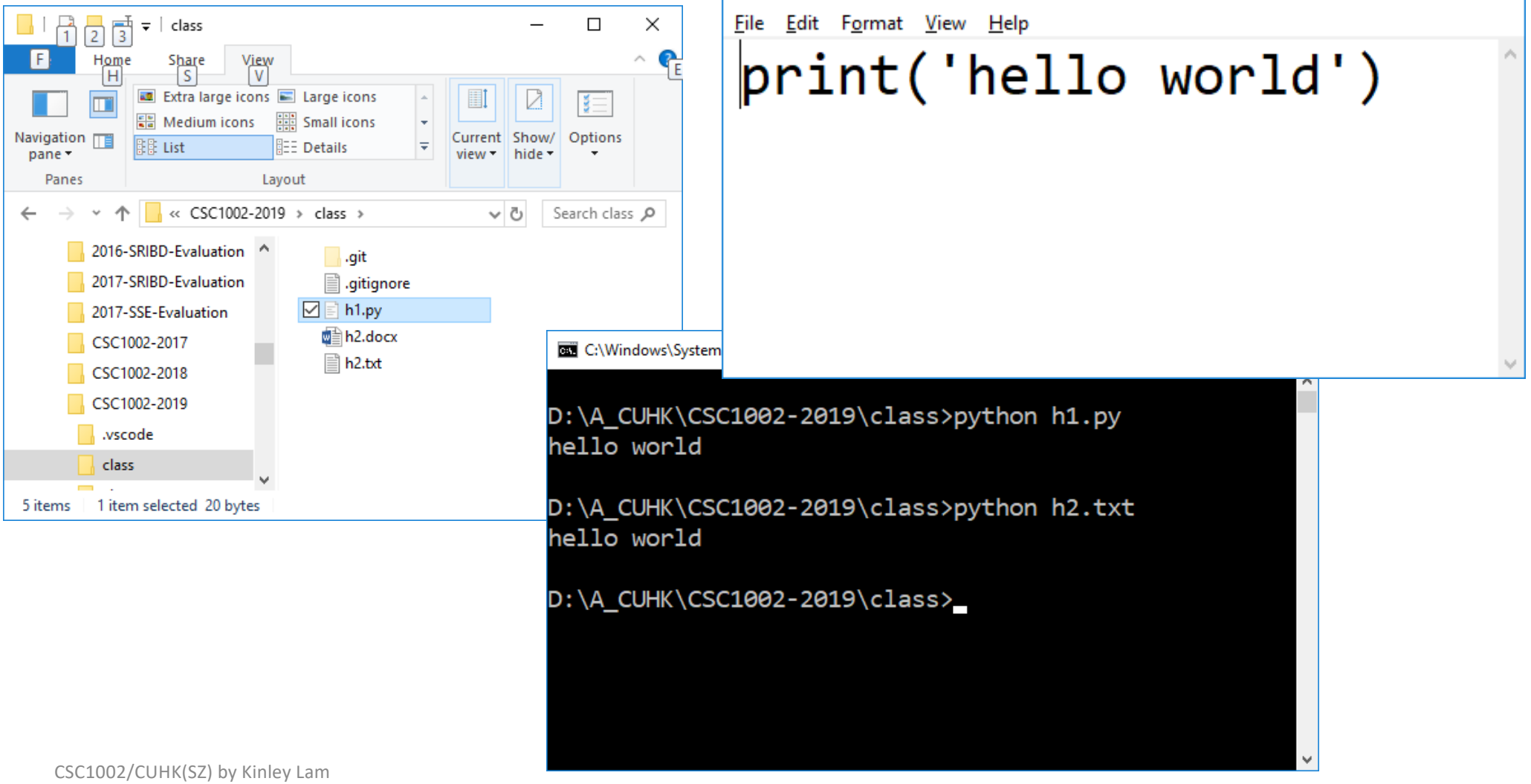
- Light weight
- Simplicity
- Cross platform (windows, macOS & linux)
- Integrated (debugging, source control, refactoring, auto code completion)
- Universal (python, C/C++, C#, html, java, swift, php, SQL …)
- Lots of extensions !!!!!
- FREE and back by Microsoft !!!

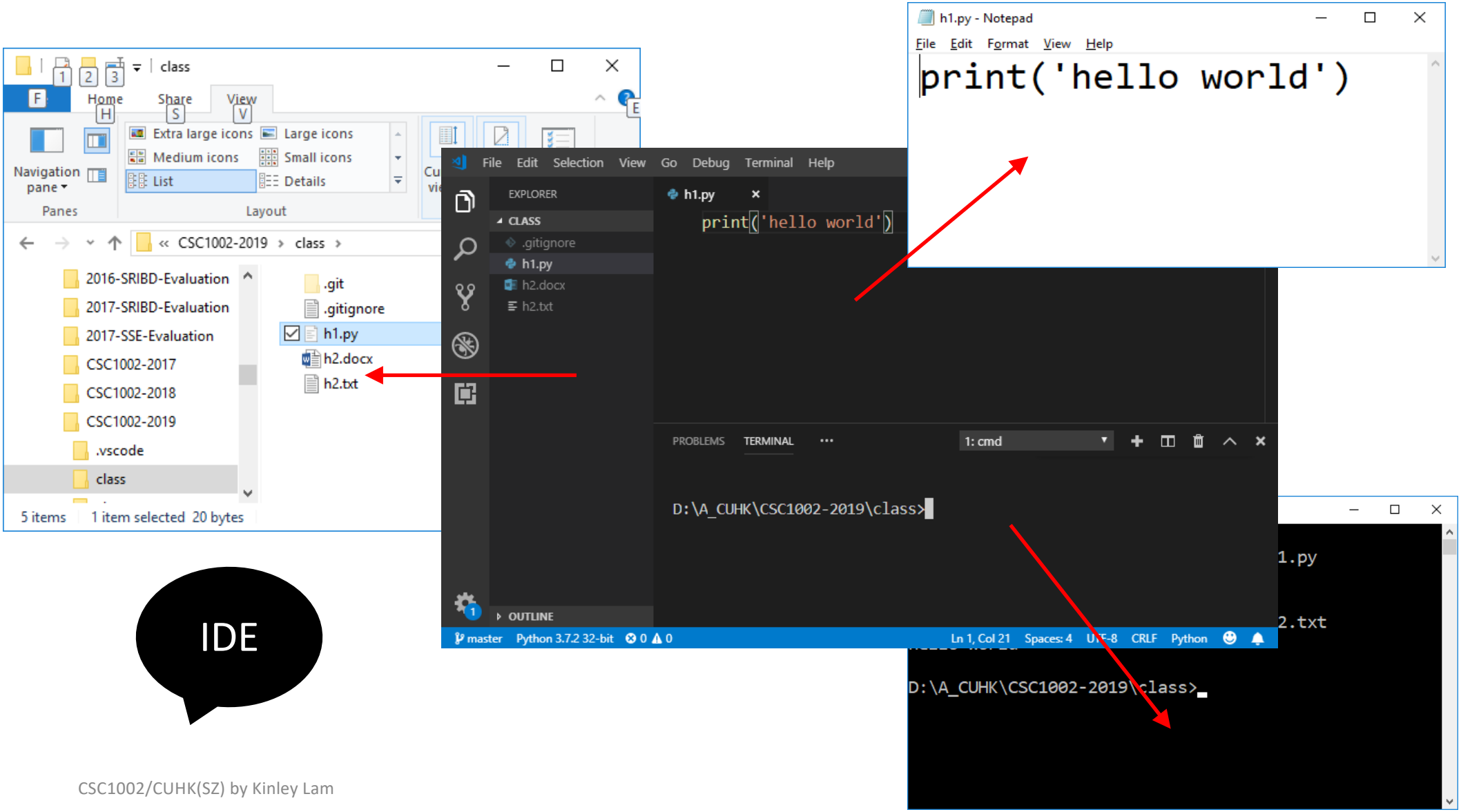
# Specialized editors





# IDE – what is it?

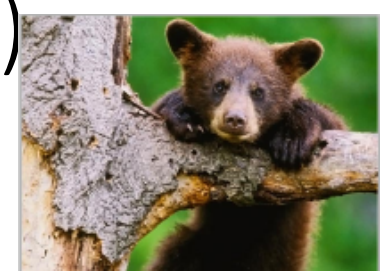

class
Home
Share
View
Extra large icons
Large icons
Medium icons
Small icons
List
Details
Navigation pane
Panes
Layout
Current view
Show/ hide
Options
« CSC1002-2019 › class ›
Search class
2016-SRIBD-Evaluation
2017-SRIBD-Evaluation
2017-SSE-Evaluation
CSC1002-2017
CSC1002-2018
CSC1002-2019
.vscode
class
.git
.gitignore
h1.py
h2.docx
h2.txt
5 items
1 item selected 20 bytes
h1.py - Notepad
File Edit Format View Help
print('hello world')
C:\Windows\System
D:\A_CUHK\CSC1002-2019\class>python h1.py
hello world
D:\A_CUHK\CSC1002-2019\class>python h2.txt
hello world
D:\A_CUHK\CSC1002-2019\class>


class
Home
Share
View
Extra large icons
Large icons
Medium icons
Small icons
List
Details
Navigation pane
Panes
Layout
« CSC1002-2019 › class ›
2016-SRIBD-Evaluation
2017-SRIBD-Evaluation
2017-SSE-Evaluation
CSC1002-2017
CSC1002-2018
CSC1002-2019
.vscode
class
.git
.gitignore
h1.py
h2.docx
h2.txt
5 items
1 item selected 20 bytes
h1.py - Notepad
File Edit Format View Help
print('hello world')
File Edit Selection View Go Debug Terminal Help
EXPLORER
CLASS
.gitignore
h1.py
h2.docx
h2.txt
print('hello world')
PROBLEMS
TERMINAL
1: cmd
D:\A_CUHK\CSC1002-2019\class>
OUTLINE
master
Python 3.7.2 32-bit
Ln 1, Col 21
Spaces: 4
CRLF
Python
1.py
2.txt
D:\A_CUHK\CSC1002-2019\class>

IDE

# Past Assignments

# Past Assignments (2017-2018)



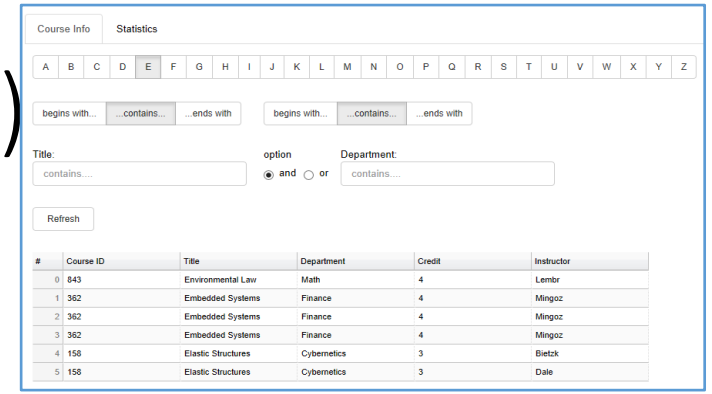
Belowarelinkstoanumberofcreativeprogrammingassignments

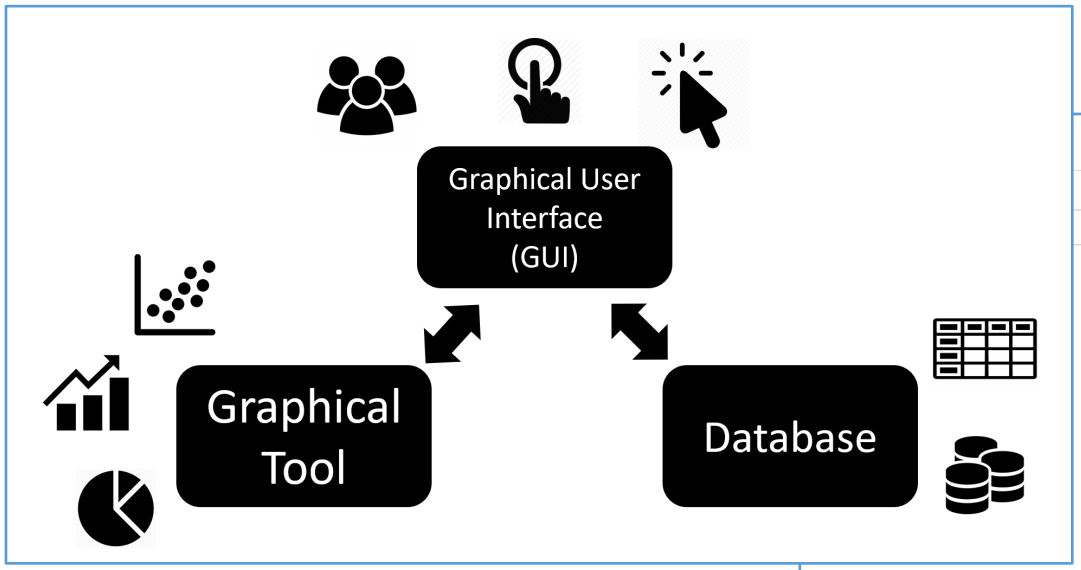
Below are links to a number of creative programming assignments

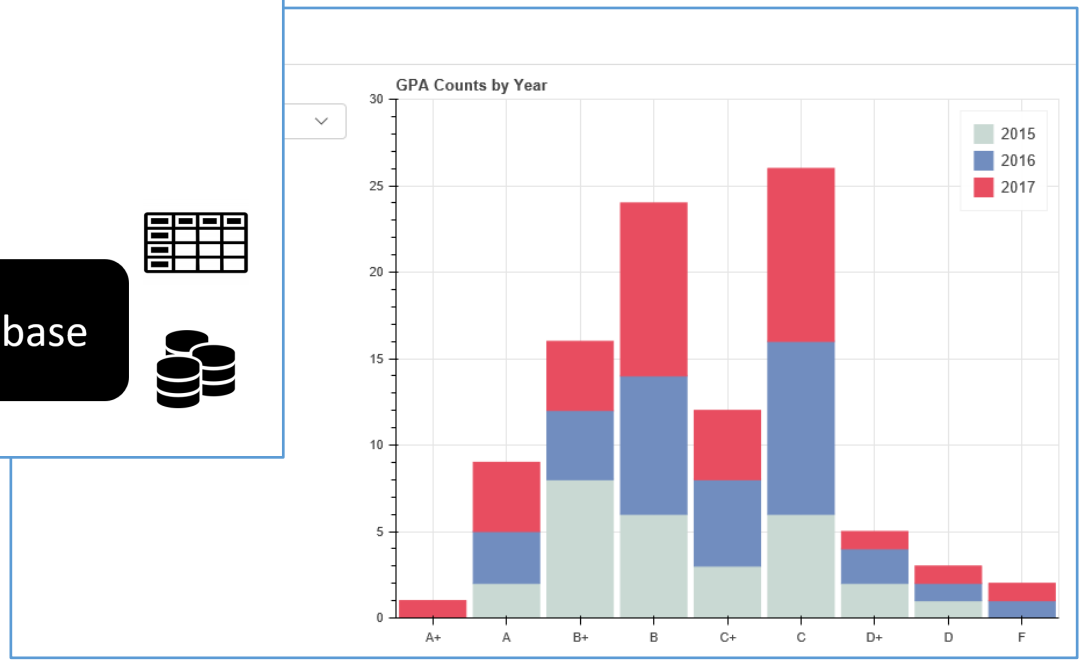
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| J | A | N | S | N | Y | E | P | A | R | G | X |
| I | P | V | O | O | R | A | N | G | E | V | T |
| W | P | K | V | L | R | P | M | E | C | U | A |
| I | L | U | E | R | E | M | A | A | M | N | G |
| K | E | M | D | U | B | M | I | P | A | I | A |
| S | O | O | U | S | E | Y | R | N | A | G | L |
| N | U | J | Y | M | U | M | A | E | M | Y | R |
| P | P | B | E | L | L | B | I | J | T | H | A |
| P | V | G | N | N | B | M | J | F | H | A | W |
| J | X | H | O | F | A | M | S | X | R | M | W |
| D | P | W | I | P | P | X | F | G | Z | Z | P |
| M | E | B | Q | W | Z | C | M | M | S | J | L |

```
Your Guess 6/8 : 6750
1234 : Correct: 0 Position : 0
5678 : Correct: 1 Position : 2
0567 : Correct: 0 Position : 4
5076 : Correct: 0 Position : 4
7650 : Correct: 2 Position : 2
6750 : Correct: 0 Position : 4

Your Guess 7/8 : 7605
1234 : Correct: 0 Position : 0
5678 : Correct: 1 Position : 2
0567 : Correct: 0 Position : 4
5076 : Correct: 0 Position : 4
7650 : Correct: 2 Position : 2
6750 : Correct: 0 Position : 4
7605 : Correct: 4 Position : 0
Congratulations !! Number : 7605
Game? (y/n) >y
```

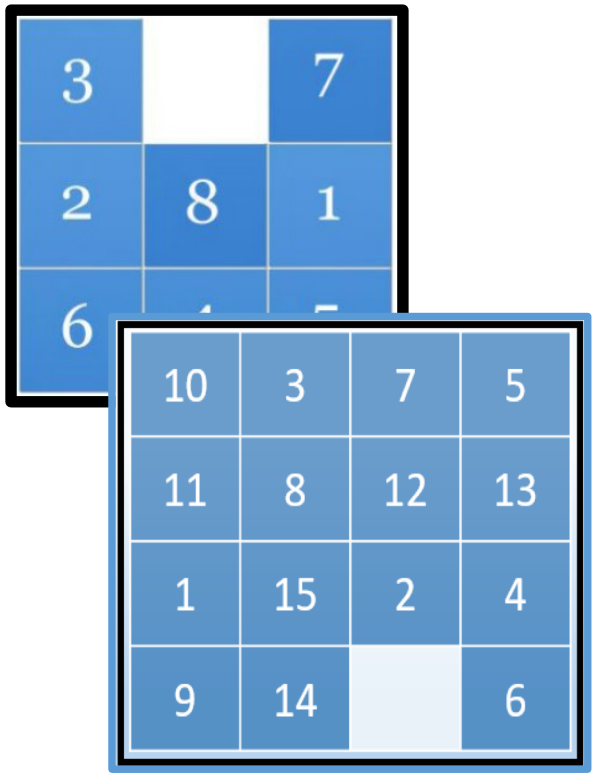
# Past Assignment (2018-2019)

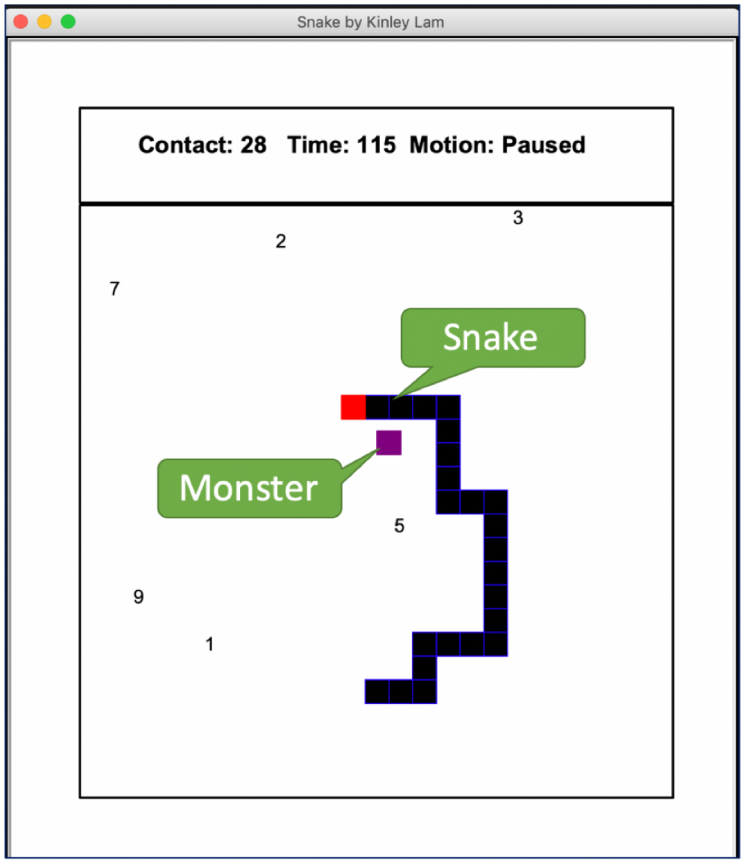


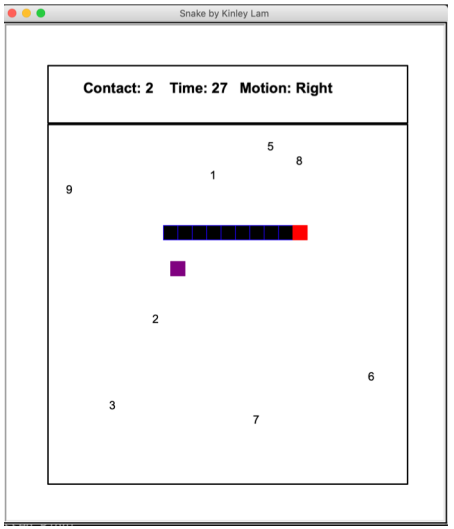



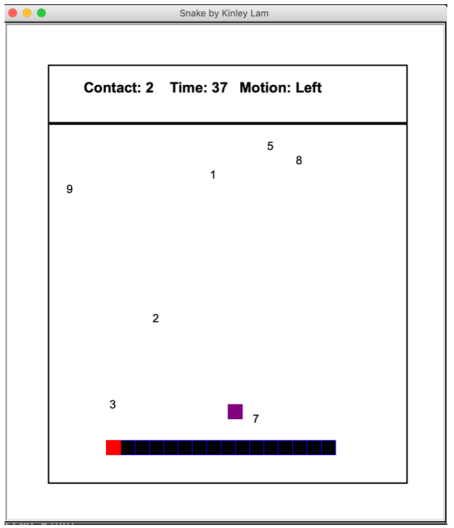



# Past Assignments (2020-2021)

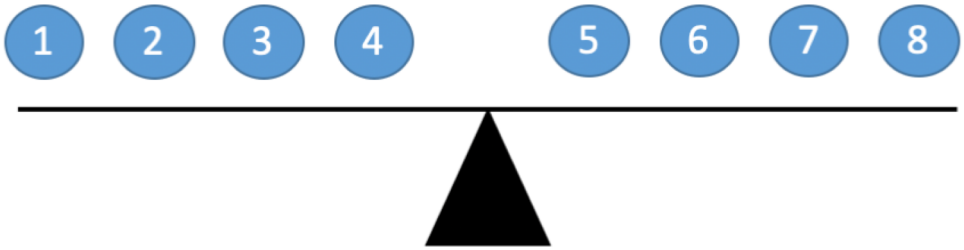


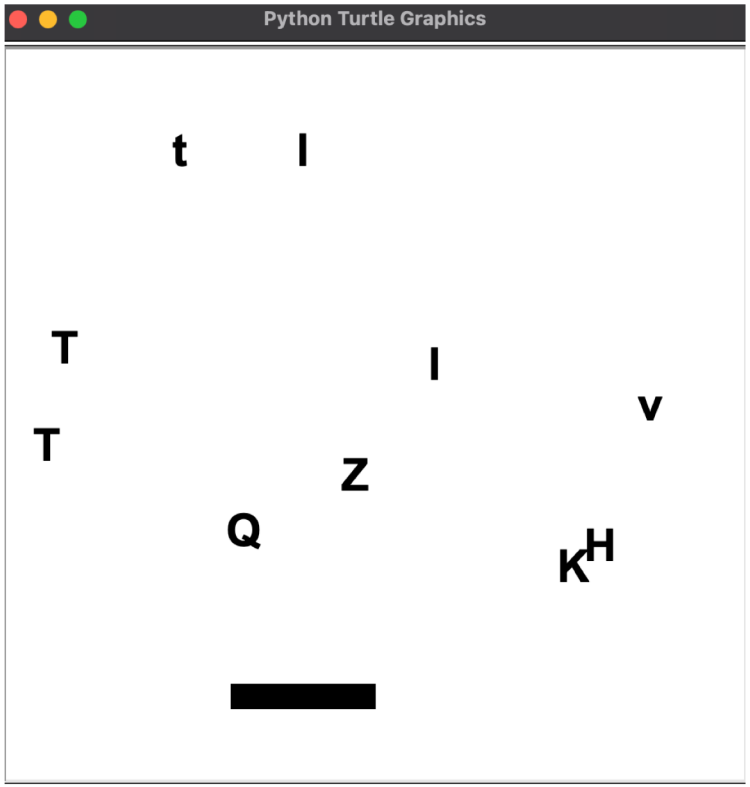



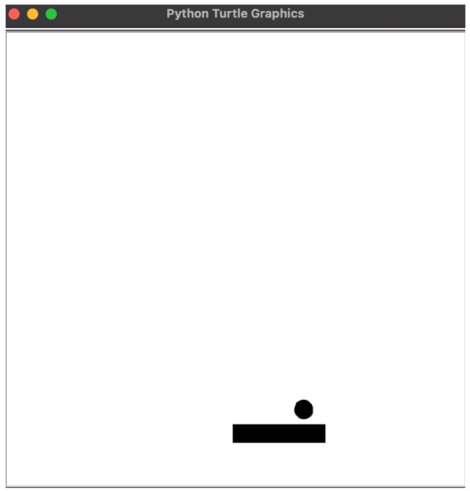



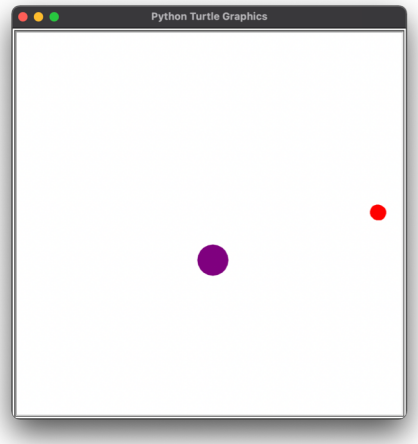
# Past Assignments (2021-2022)

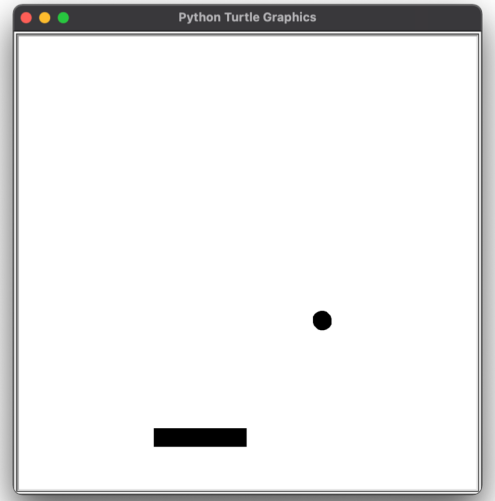



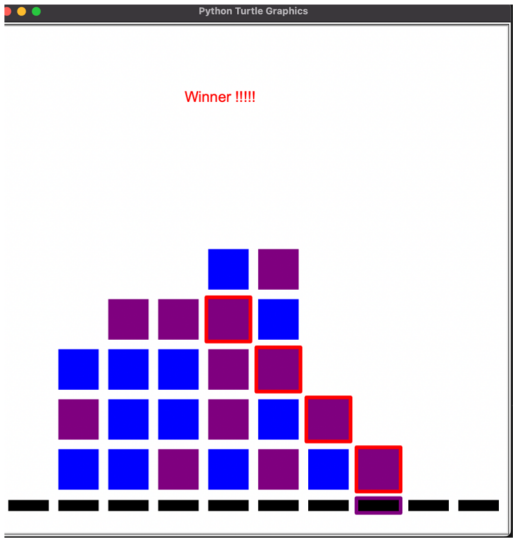
# This Year ???


Python Turtle Graphics
t
I
T
I
v
T
Z
Q
K
H


Python Turtle Graphics


Python Turtle Graphics


Python Turtle Graphics


Python Turtle Graphics
Winner !!!!!

# Teaching Plan & Assessment

Please refer to BB for more info !!!

# Python overview

# modules – reload

```
>>> import helloworld
hello world!
>>>
>>> import helloworld
>>> import helloworld
>>>
>>>
```

```
>>> from imp import reload
>>> reload(helloworld)
hello world!
```

# __name__ == “__main__”

```
def helloworld():
    print('hello world!')

if __name__ == "__main__":
    helloworld()
```

```
>>>
>>> import helloworld
>>> helloworld.helloworld()
hello world!
>>> from helloworld import helloworld as hw
>>> hw()
hello world!
>>>
```

# Core Data Types

| | |
|---|---|
| Numbers | `1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction()` |
| Strings | `'spam', "Bob's", b'a\x01c', u'sp\xc4m'` |
| Lists | `[1, [2, 'three'], 4.5], list(range(10))` |
| Dictionaries | `{'food': 'spam', 'taste': 'yum'}, dict(hours=10)` |
| Tuples | `(1, 'spam', 4, 'U'), tuple('spam'), namedtuple` |
| Files | `open('eggs.txt'), open(r'C:\ham.bin', 'wb')` |
| Sets | `set('abc'), {'a', 'b', 'c'}` |
| Other core types | Booleans, types, None |

# python – quick overview

- Indentation

```
for i in [1, 2, 3, 4, 5]:
    print i
    for j in [1, 2, 3, 4, 5]:
        print j
        print i + j
    print i
print "done looping"
```

```
# first line in "for i" block

# first line in "for j" block
# last line in "for j" block
# last line in "for i" block
```

- Modules

```
import re
my_regex = re.compile("[0-9]+", re.I)
```

```
import re as regex
my_regex = regex.compile("[0-9]+", regex.I)
```

# python – quick overview

- Strings

```
single_quoted_string = 'data science'
double_quoted_string = "data science"

multi_line_string = """This is the first line.
and this is the second line
and this is the third line"""
```

- Lists

```
integer_list = [1, 2, 3]
heterogeneous_list = ["string", 0.1, True]
list_of_lists = [ integer_list, heterogeneous_list, [] ]

list_length = len(integer_list)     # equals 3
list_sum    = sum(integer_list)     # equals 6

1 in [1, 2, 3]    # True
0 in [1, 2, 3]    # False
```

# python – quick overview

- Tuples (immutable list)
  - works exactly as list except that tuples are read only and parentheses “()” are used instead of square brackets “[]”

```
>>> T = (1,2,3,4)
>>> T[1] = 88
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

- Dictionary

```
tweet = {
    "user" : "joelgrus",
    "text" : "Data Science is Awesome",
    "retweet_count" : 100,
    "hashtags" : ["#data", "#science", "#datascience", "#awesome", "#yolo"]
}
```

# python – quick overview

- defaultdict() – 4 x word-count implementations

A

```
word_counts = {}
for word in document:
    if word in word_counts:
        word_counts[word] += 1
    else:
        word_counts[word] = 1
```

B

```
word_counts = {}
for word in document:
    try:
        word_counts[word] += 1
    except KeyError:
        word_counts[word] = 1
```

C

```
word_counts = {}
for word in document:
    previous_count = word_counts.get(word, 0)
    word_counts[word] = previous_count + 1
```

D

```
word_counts = defaultdict(int)
for word in document:
    word_counts[word] += 1
```

# python – quick overview

- defaultdict() – list, dict and function

```
dd_list = defaultdict(list)
dd_list[2].append(1)

dd_dict = defaultdict(dict)
dd_dict["Joel"]["City"] = "Seattle"

dd_pair = defaultdict(lambda: [0, 0])
dd_pair[2][1] = 1
```

# python – quick overview

- Control Flow - if

```
if 1 > 2:
    message = "if only 1 were greater than two..."
elif 1 > 3:
    message = "elif stands for 'else if'"
else:
    message = "when all else fails use else (if you want to)"

parity = "even" if x % 2 == 0 else "odd"
```

- Control Flow - While

```
x = 0
while x < 10:
    print x, "is less than 10"
    x += 1
```

# python – quick overview

- Control Flow – for

```
for x in range(10):
    print x, "is less than 10"
```

- List Comprehensions

```
even_numbers = [x for x in range(5) if x % 2 == 0]  # [0, 2, 4]
squares      = [x * x for x in range(5)]            # [0, 1, 4, 9, 16]
even_squares = [x * x for x in even_numbers]        # [0, 4, 16]

square_dict = { x : x * x for x in range(5) }  # { 0:0, 1:1, 2:4, 3:9, 4:16 }
square_set  = { x * x for x in [1, -1] }       # { 1 }

pairs = [(x, y)
         for x in range(10)
         for y in range(10)]   # 100 pairs (0,0) (0,1) ... (9,8), (9,9)
```

# python – quick overview

- Functional Tools – partial, map & reduce

```
9   from functools import partial, reduce
10
11  def exp(base, power):
12      return base ** power
13
14  base2 = partial(exp,2)
15  base10 = partial(exp,10)
16
17  print(base2(3))
18  print(base10(3))
19
20  print(list(map(base2, [1,2,3,4])))
21  print(list(map(base10, [1,2,3,4])))
22
23  def multiply(x,y):
24      return x*y
25
26  print(reduce(exp, [1,2,3,4]))
27  print(reduce(multiply, [1,2,3,4]))
```

Output

```
8

1000

[2, 4, 8, 16]

[10, 100, 1000, 10000]

1

24
```

# python – quick overview

- zip & unpacking

```
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
zip(list1, list2)        # is [('a', 1), ('b', 2), ('c', 3)]

pairs = [('a', 1), ('b', 2), ('c', 3)]
letters, numbers = zip(*pairs)
```

# python – quick overview

- args & kwargs

```
def magic(*args, **kwargs):
    print "unnamed args:", args
    print "keyword args:", kwargs

magic(1, 2, key="word", key2="word2")
```

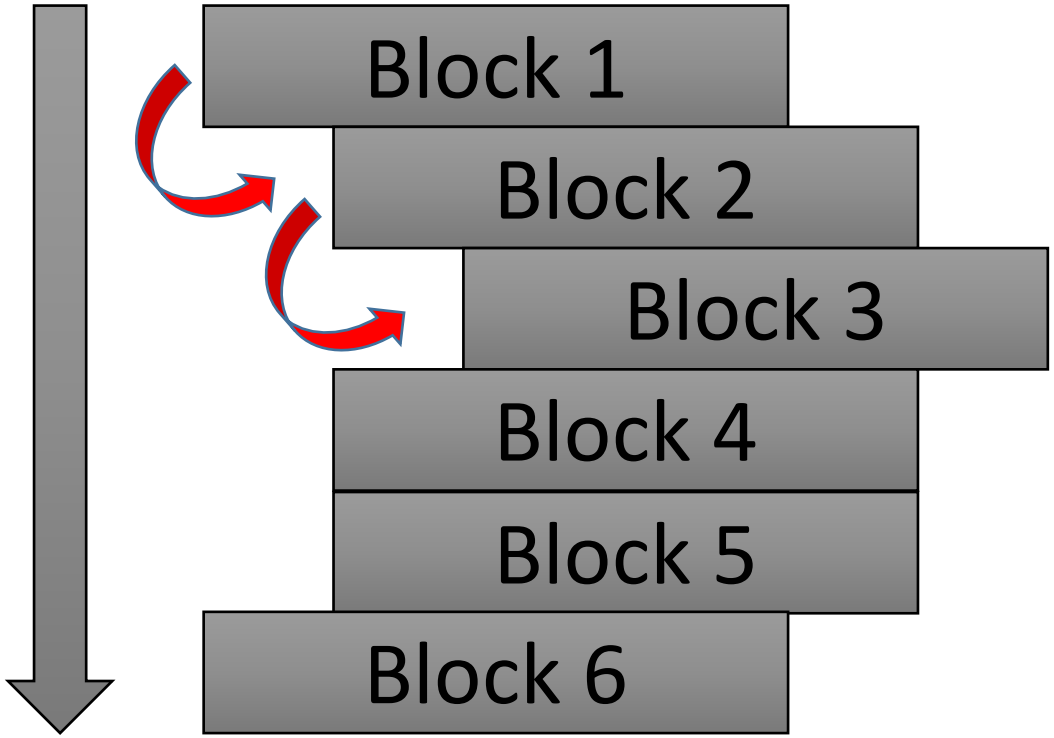
- args = tuple of unnamed args,
- kwargs = dictionary list of named args

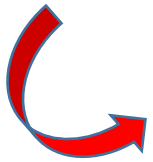
```
def other_way_magic(x, y, z):
    return x + y + z

x_y_list = [1, 2]
z_dict = { "z" : 3 }
print other_way_magic(*x_y_list, **z_dict)   # 6
```

# indentation – code blocks

- unlike others (C, C++, Pascal, C#, Javascript, VB, … and so on),
- no curly braces, no explicit begin/end or keywords to mark where block starts and stops
- python uses “Indentation” to group block of source codes together as logical blocks that are conditionally executed together with controls
- spaces or tab can be used to create the indentation (usually 4 spaces)
- python uses “:” to mark beginning of a block (def, if, for, while …)
- goals: consistent, uniform, readable styles

# execution sequence



conditional
or controls

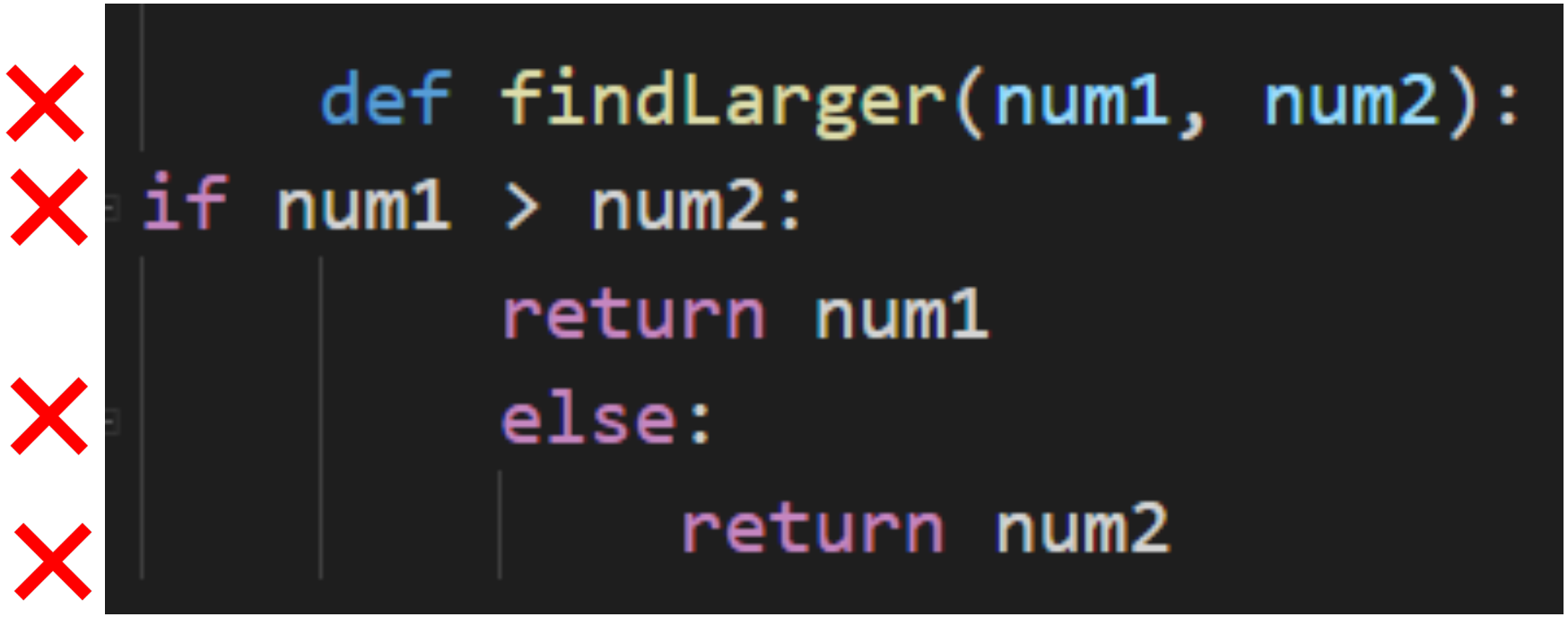
```
def promptNumber(desc):
    while True:
        n1=input('enter ' + desc + ' number >')
        if n1.isdigit():
            n1=int(n1)
            nums.append(n1)
            break
```

```
def promptNumber(desc):
    while True:
        n1=input('enter ' + desc + ' number >')
        if n1.isdigit():
            n1=int(n1)
            nums.append(n1)
            break
```

?

```
print('hello world')
    print('hello world')
```

```
    def findLarger(num1, num2):
if num1 > num2:
        return num1
        else:
            return num2
```


def findLarger(num1, num2):
if num1 > num2:
return num1
else:
return num2

# C/C++

- first programmer:

```
while (x>y) {
    ……;
    ……;
}
```

- 2nd programmer:

```
while (x>y) {
    ……;
    ……;
                ……;
                ……;
}
```

- 3rd programmer:

```
while (x>y) {
    ……;
    ……;
                ……;
                ……;
…….;
……;
}
```

- 4th programmer:

```
while (x>y) {
    ……;
    ……;
……;
                ……;
                                    …….;
……;
}
```

# C/C++ - classic pitfall


```
if (x)
     if (y)
          statement1;
else
     statement2;
```

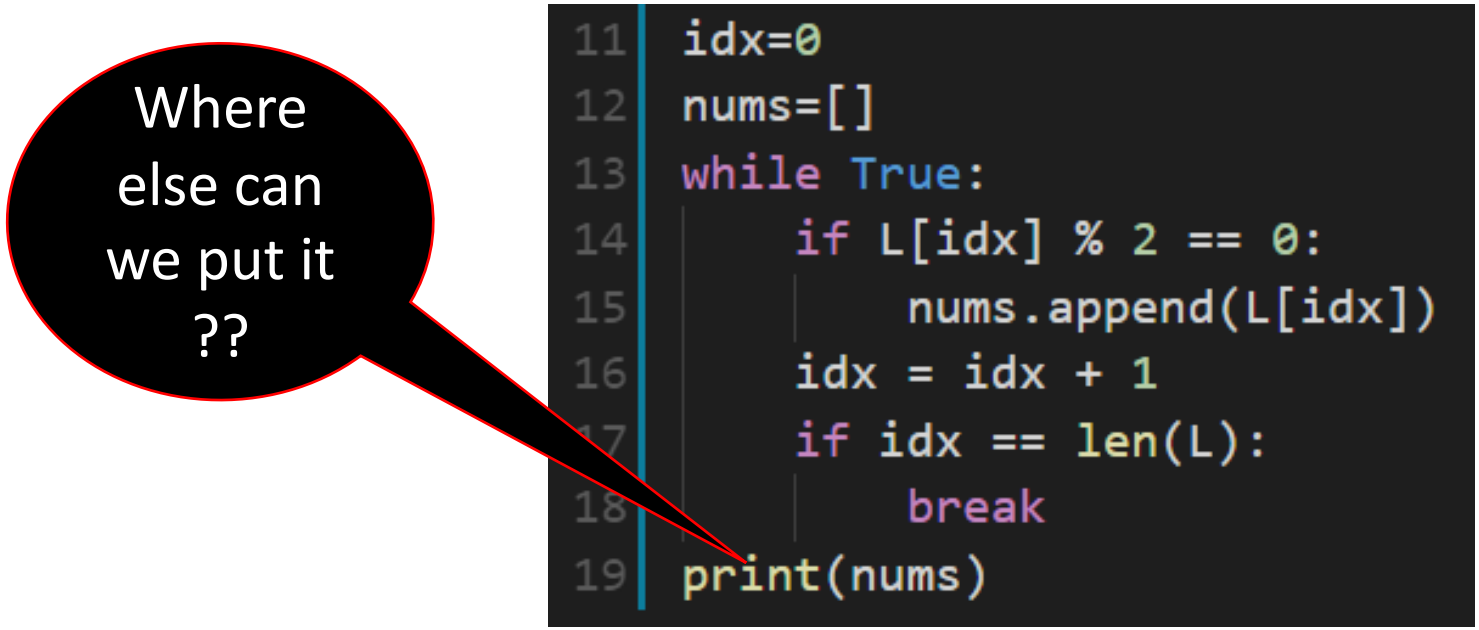
# python – if/else


```
if x:
     if y:
          statement1
else:
     statement2
```

# List Iterations

# while, for & list comprehension

- Given L = [3,45,22,44,78,90,44,23], extract all even numbers into another list



How to show the list incrementally ??

```
21  idx=0
22  nums=[]
23  while idx < len(L):
24      if L[idx] % 2 == 0:
            nums.append(L[idx])
26      idx += 1
27  print(nums)
```

# “for” vs list comprehension

```
nums=[]
for num in L:
    if num % 2 == 0:
        nums.append(num)
print(nums)
```

```
35 nums = [x for x in L if x % 2 == 0]
36 print(nums)
```

# controls – for (sequence assignment)

- Tuple

```
T = [ (1,2), (2,3), (3,4), (4,5) ]

for x in T:
    second = x[1]
    print(second)


# second element of each tuple
for (f,s) in T:
    print(s)
```

```
Tuple >
2
3
4
5
```

# controls – for (sequence assignment)

- List (extracting the first and last item from each sub list)

```
L = [ [1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15] ]

for (head, *body, tail) in L:
    print(head,tail)
```

```
List >
1 5
6 10
11 15
```

# control – ternary if

```
if cond:
    ret = val1
else:
    ret = val2
```

```
ret = val1 if cond else val2
```

```python
def findLarger(num1, num2):
    if num1 > num2:
        return num1
    else:
        return num2
```

```python
def findLarger(num1, num2):
    return num1 if num1 > num2 else num2
```

# String Formatting

# string formatting – 2 variations

- Expression: ‘ …..%…….  ’ % (values)
- Method: ‘ ……{}…… ‘.format(values)
- ‘Do you like %s or %s?’ % ( ‘apples’, ‘oranges’ )
- ‘Do you like {} or {}?’.format(‘apples’, ‘oranges’)
- ‘Do you like apples or oranges?’
- Note: both pretty much achieve same results, the format method was introduced later and gains more popularity.

# method-call formatting – position

- relative position

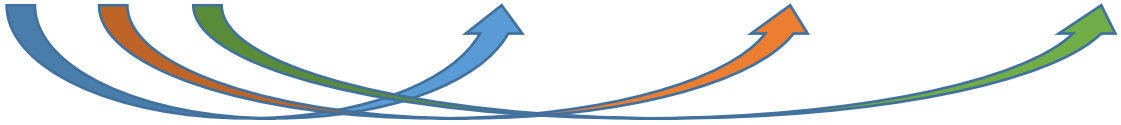
  ‘{} {} {}’.format(‘apple’,’orange’,’grape’)

- position number

  ‘{0} {1} {2}’.format(‘apple’,’orange’,’grape’)

- keyword

  ‘{a} {o} {g}’.format( a=‘apple’, o=‘orange’, g=‘grape’ )



# example 1

- input
  - a = ‘apple’, o = ‘orange’, g = ‘grape’
- output
  - ‘apple-orange-apple-orange-apple-grape-orange’
- using print() statement
  - print(a + ‘-’ + o + ‘-’ + a + ‘-’ + o + ‘-’ + a + ‘-’ + g + ‘-’ + o)
- using format()
  - print(‘{0}-{1}-{0}-{1}-{0}-{2}-{1}’.format(a,o,g))
  - print(‘{a}-{o}-{a}-{o}-{a}-{g}-{o}’.format(‘a’=a,’o’=o,’g’=g))

# example 2

- input
  - total = 4356, failed=342
- output
  - ‘Total 4356 samples with failures 342 cases, failure rate 12.76%’
- using print() statement

  ‘Total ’ + str(total) + ‘ samples with failures ‘ + str(failed) + ‘ cases, failure rate ‘ + round(total/failed,2) + ‘%’
- using format()

  ‘Total {} samples with failures {} cases, failure rate {}%’.

  format(total, failed, round(total/failed,2))

# Separate format string from data

- output
  - ‘Total {} samples with failures {} cases, failure rate {}%’
  - ‘共有{}件样品失败{}件，失败率{}%’
  - ‘총 {} 건의 실패 사례 {} 건, 실패율 {}%’
  - 'Total {} muestras con fallas {} casos, índice de fallas {} %'

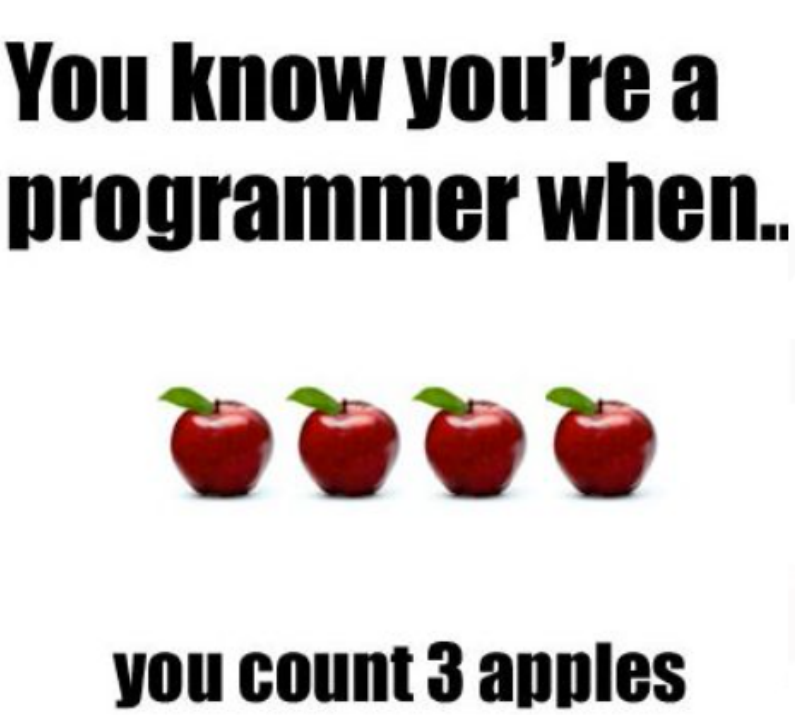
# string at a glance

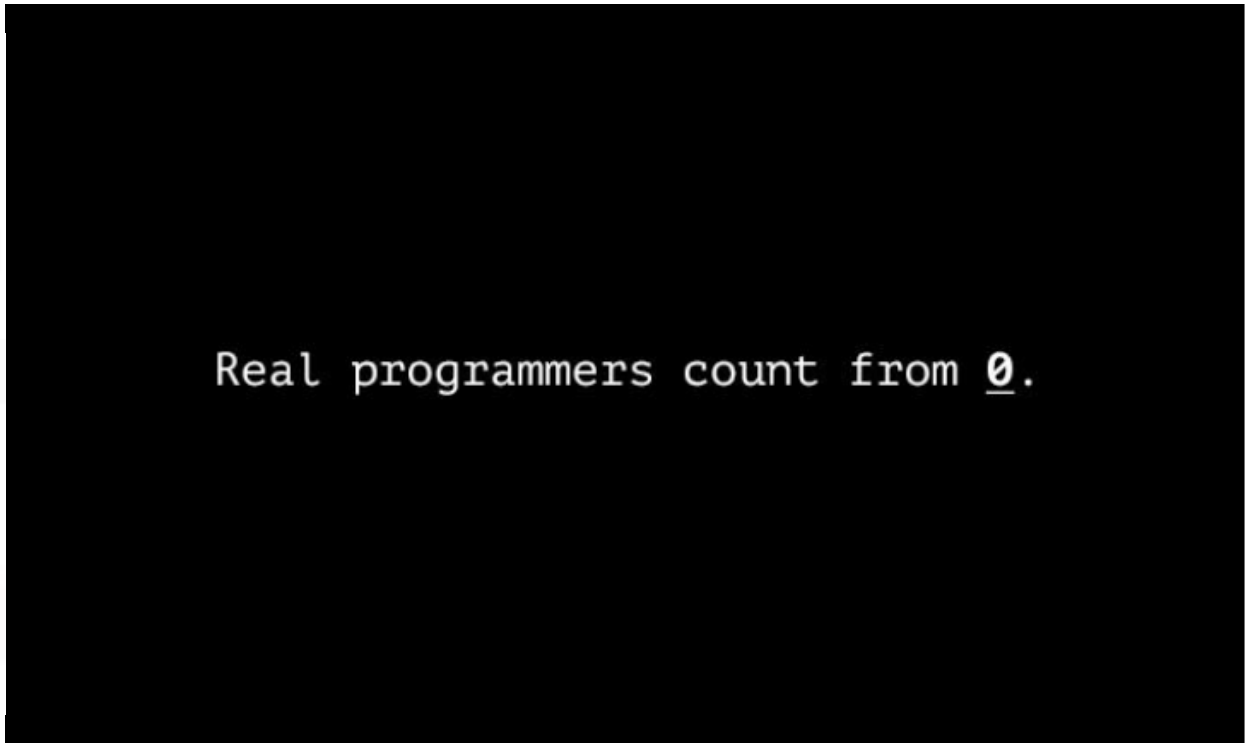
| | |
|---|---|
| `S = ''` | Empty string |
| `S = "spam's"` | Double quotes, same as single |
| `S = 's\np\ta\x00m'` | Escape sequences |
| `S = """...multiline..."""` | Triple-quoted block strings |
| `S = r'\temp\spam'` | Raw strings (no escapes) |
| `S.isdigit()` | content test, |
| `S.lower()` | case conversion, |
| `S.endswith('spam')` | end test, |
| `'spam'.join(strlist)` | delimiter join, |
| `for x in S: print(x)` | Iteration, membership |
| `'spam' in S` | |
| `[c * 2 for c in S]` | |
| `S1 + S2` | Concatenate, repeat |
| `S * 3` | |
| `S[i]` | Index, slice, length |
| `S[i:j]` | |
| `len(S)` | |
| `"a %s parrot" % kind` | String formatting expression |
| `"a {0} parrot".format(kind)` | String formatting method in 2.6, 2.7, and 3.X |
| `S.find('pa')` | String methods (see ahead for all 43): search, |
| `S.rstrip()` | remove whitespace, |
| `S.replace('pa', 'xx')` | replacement, |
| `S.split(',')` | split on delimiter, |

# strings

- ‘this is a string single quoted’
- “this is another string’s double quoted”
- ‘’’this is another triple quoted string’’’
  - spanning multiple lines
- r”c:\dir1\dir2\dir3\file.txt”
  - instead of “c:\\dir1\\dir2\\dir3\\file.txt”
- byte and unicode strings (we will skip these for this class)
- ‘string1’ “string2” ‘string3’ (a concatenated string)
- ‘string1’, “string2”, ‘string3’ (a tuple)
- strings are immutable (cannot be changed)

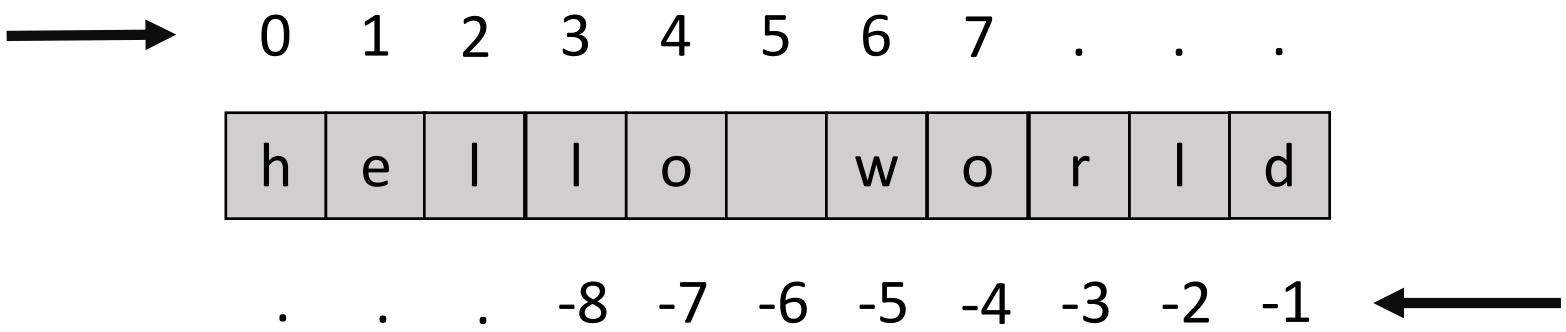
# zero-based indexing



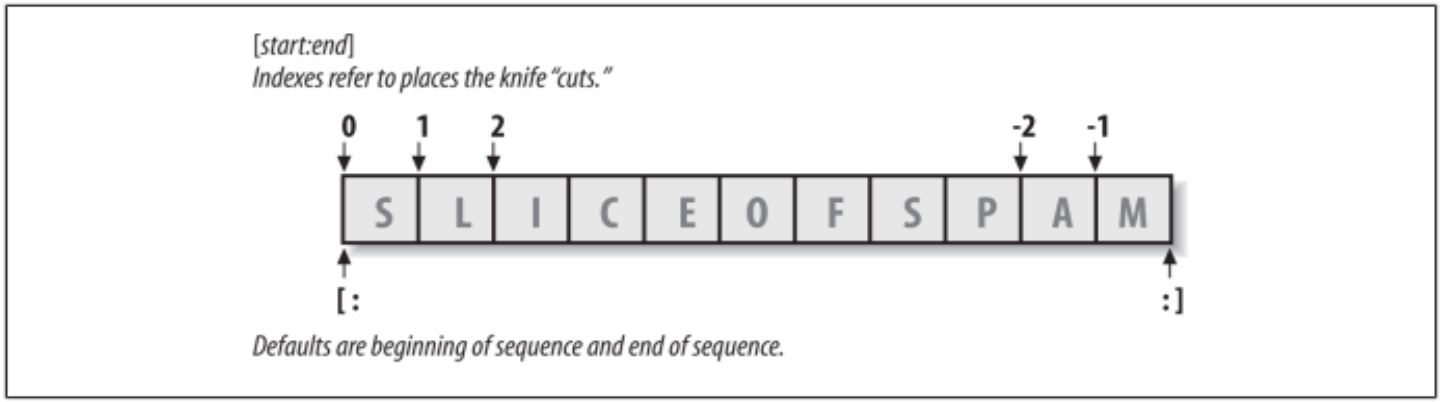
# string : indexing

- given s = “hello world”




- use module “string” and try indexing with “ascii_lowercase”
- s = string.ascii_lowercase
- indexing: s[0], s[1], s[-1], s[-2], s[26]

# string : slicing




- use module “string” and try indexing and slicing with “ascii_lowercase”
- s = string.ascii_lowercase
- slicing: s[:], s[1:], s[:-1], s[1:-1], s[-6:], s[::2], s[::-1], s[1:10:-1]
- how to reverse a string using slicing? s[::-1]

# string – common functions

- justification & conversion
  - center(), ljust(), rjust(), format(), capitalize(), lower(), upper()
- search & replace
  - endswith(), startswith(), find(), rfind(), replace()
- query
  - isalnum(), isalpha(), isdecimal(), isdigit(), islower(), isnumeric(), isupper(), isspace()
- splitting and joining
  - join(), split(), rsplit()
- stripping whitespaces
  - strip(), lstrip(), rstrip()

# string - exercises

- ‘apple’.center(10), ‘apple’.rjust(10), ‘apple’.ljust(10), ‘apple’.upper()
- ‘apple’.startswith(‘ap’), ‘apple’.endswith(‘le’), ‘apple’.find(‘app’)
- ‘apple’.replace(‘app’,’peop’), ‘apple’.islower(), ‘apple’.isnumeric()
- ‘apple’.isalnum(), ‘apple’.isdigit(), ‘apple’.isdelicious()
- try to generate the following figure using series of print statements:

- print statements
  - print( (‘p’).center(11) )
  - print( (‘p’*3).center(11) )
  - print( (‘p’*5).center(11) )
  - print( (‘p’*7).center(11) )
  - print( (‘p’*9).center(11) )
  - print( (‘p’*11).center(11) )
- ‘apple-orange-grape’.split(‘-’)
- ‘apple-orange-grape’.partition(‘-’)
- Note: split() splits all occurrences, while partition() only on first occurrence.
- ‘-’.join([‘apple’, ‘orange’, ‘grape’])

# range() & list()

- range() is a  handy function to generate a sequence of successive integers
- list() is used to convert the numbers from range() into a list
- range(start, end, step)
- try:
  - list(range(10))  [0,1,2,3,4,5,6,7,8,9]
  - list(range(1,10))  [1,2,3,4,5,6,7,8,9]
  - list(range(0,10,2))  [0,2,4,6,8]
  - list(range(1,10,2))  [1,3,5,7,9]

# string – exercises (using range())

- using range with a for-loop:

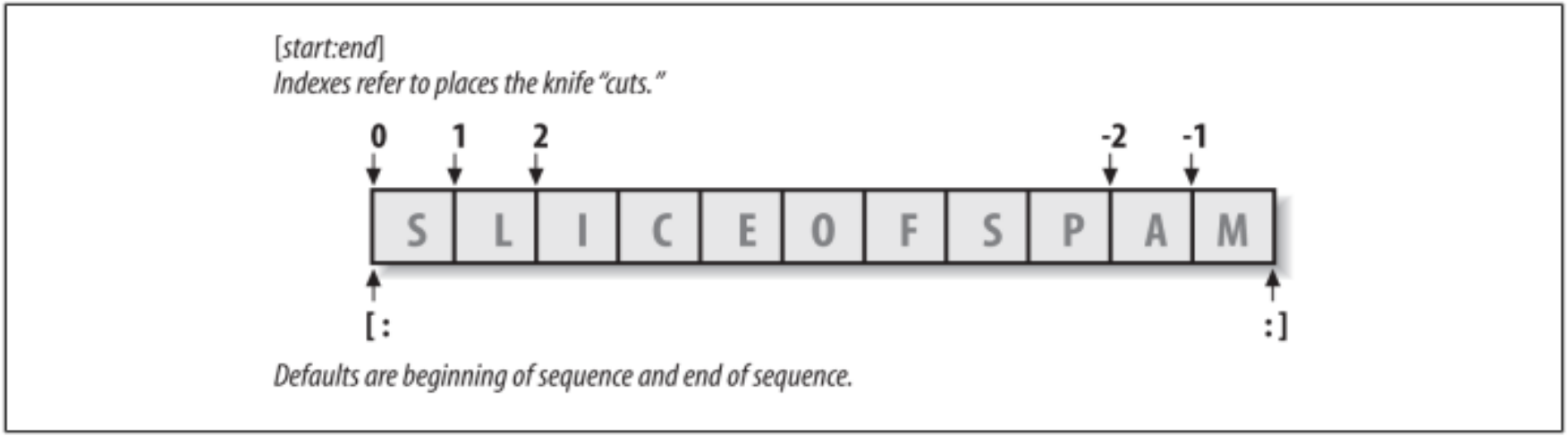

- for i in range(1,12,2): print( ('p'*i).center(11) )

# List - examples

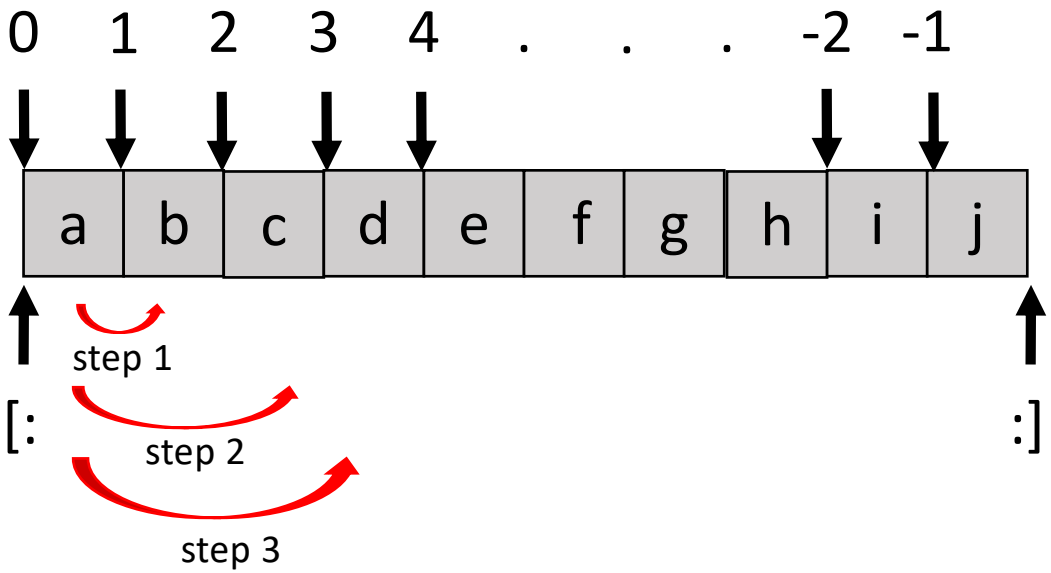
| List | Remark |
|---|---|
| [] | empty list |
| [ 12, ‘a’, 1.24, {‘d’:88}] | List with 4 items of different types |
| list(range(100)) | list of numbers from 0 to 99 |
| [ [1,2,3], [4,5,6], [7,8,9] ] | 3 x 3 matrix |

# List – slicing & indexing

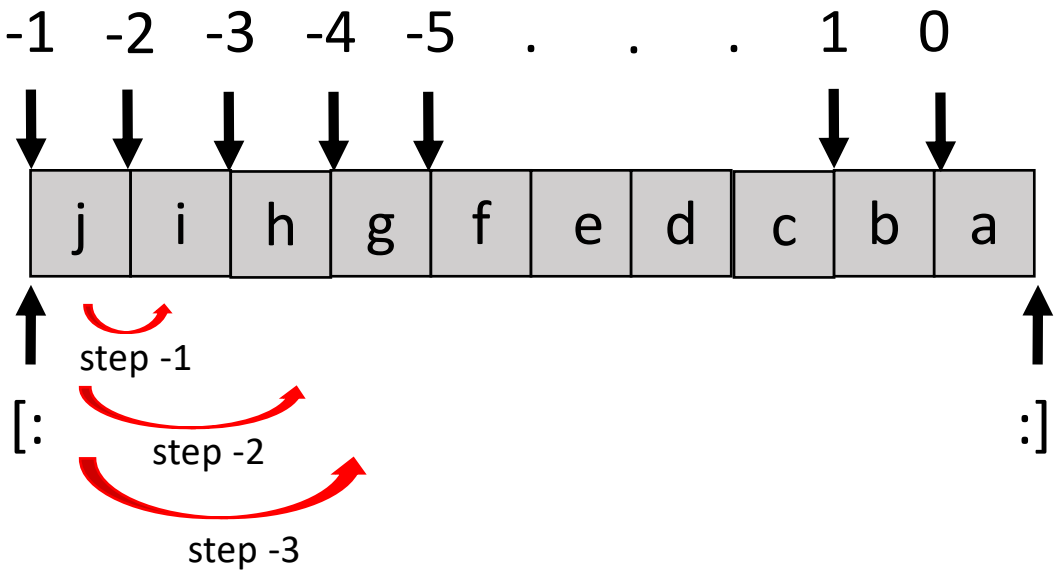
- Similar to string's



# slicing – step > 0 (given “abcdefghij”)



# slicing – step < 0 (given “abcdefghij”)



# List – update operations L=[1,2,3]

| by slicing | by method | Remark |
|---|---|---|
| L[:0] = [9] | L.insert(0,[9]) | [9,1,2,3] insert front |
| L[1:1] = [9] | L.insert(1,[9]) | [1,9,2,3] insert between |
| L[1:2] = [9], L[1] = 9 | | [1,9,3] replace |
| L[len(L):] = [9]<br>L + [9] | L.extend([9]) | [1,2,3,9] insert end |
| L[0:1] = []<br>del L[0] | L.remove(1) | [2,3] remove first item |
| L[len(L):] = [[9]] | L.append([9]) | [1,2,3, [9]] nested append |
| | L.index(1) | 0 return index of item 1 |
| L[-1:] = [] | L.pop() | [1,2] remove last item |

# list – other operations L = [1,2,3]

| | result | Remark |
|---|---|---|
| len(L) | 3 | length |
| 1 in L | true | membership |
| for x in L: print(x, end=‘,’) | 1,2,3, | iteration |
| [x*x for x in L] | [1,4,9] | list comprehension |
| L.sort(reverse=True) | [3,2,1] | sort in reverse order |

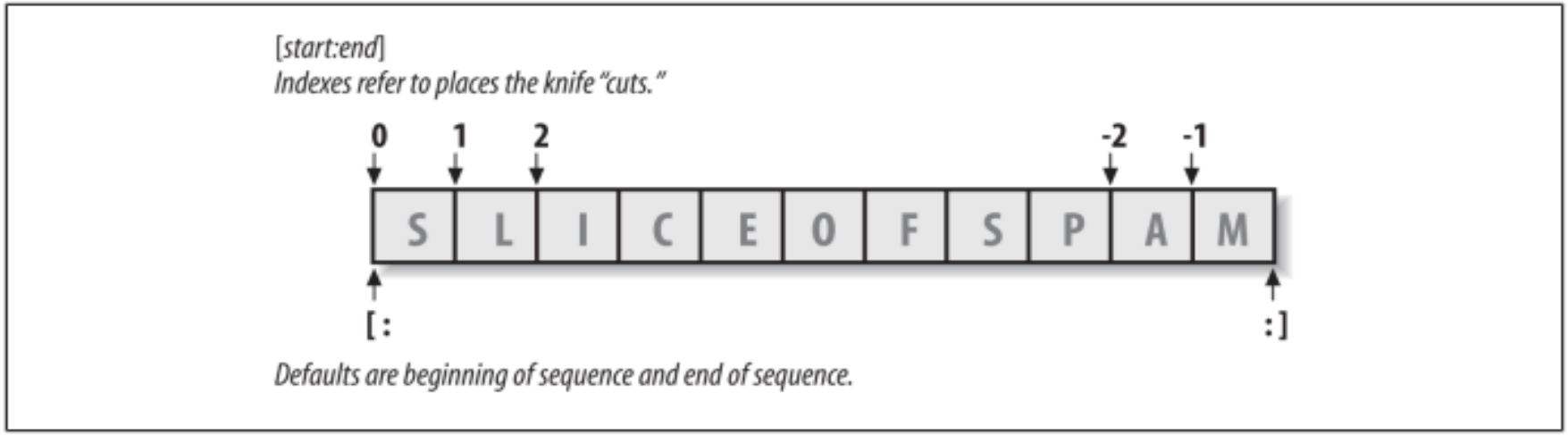
# Tuple - examples

| List | Remark |
|---|---|
| () | empty tuple |
| (12,) | single item tuple (please note comma) |
| ( 12, ‘a’, 1.24, {‘d’:88} ) | tuple with 4 items of different types |
| tuple(range(100)) | list of numbers from 0 to 99 |
| ( [1,2,3], [4,5,6], [7,8,9] ) | 3 x 3 matrix |
| list( (1,2,3,4) ) | converted to list |

# Tuple - notes

- Tuples work exactly the same as list except that tuples are immutable (read only) and parentheses “()” are used instead of square brackets “[]”
- create a tuple T = (1,2,3,4)
- type T[0]
- try assigning a new value to T[0], say T[0] = 9, what do you get?

# tuple : slicing and indexing




- similar to list's and string's
- indexing: s[0], s[1], s[-1], s[-2], s[26]
- slicing: s[:], s[1:], s[:-1], s[1:-1], s[-6:], s[::2], s[::-1], s[1:10:-1]?
- how to reverse items in a tuple using slicing? s[::-1]

# list – slicing exercise

- given s = list(range(100))
- use slicing to return all even numbers from s
  - [0,2,4,6,8,….,98]
- use slicing to return all odd numbers from s
  - [1,3,5,7,9,….,99]
- use slicing to return all odd numbers from s in reverse order
  - [99,97,95,93,91,…….,1]
- use slicing to return all even numbers from s in reverse order
  - [98,96,94,92,90,…….,0]
- use slicing to return the last 10 numbers from s in reverse order
  - [99,98,97,96,95,94,93,92,91,90]
- use slicing to return first 3 numbers from s in reverse order
  - [2,1,0]