# Lecture 2, Part 2: Programming in Python

# Course outline

- Part 1 Introduction to Computing and Programming (first 2 weeks):
  - Problem solving: Problem statement, algorithm design, programming, testing, debugging
  - Scalar data types: integers, floating point, Boolean, others (letters, colours)
  - Arithmetic, relational, and logical operators, and expressions
  - Data representation of integers, floating point, Boolean
  - Composite data structures: string, tuple, list, dictionary, array
  - Sample operations on string, tuple, list, dictionary, array
  - Algorithms (written in pseudo code) vs. programs
  - Variables and constants (literals): association of names with data objects
  - A language to write pseudo code
  - Programming languages: compiled vs. interpreted programming languages
  - Python as a programming language
  - Computer organization: processor, volatile and non-volatile memory, I/O

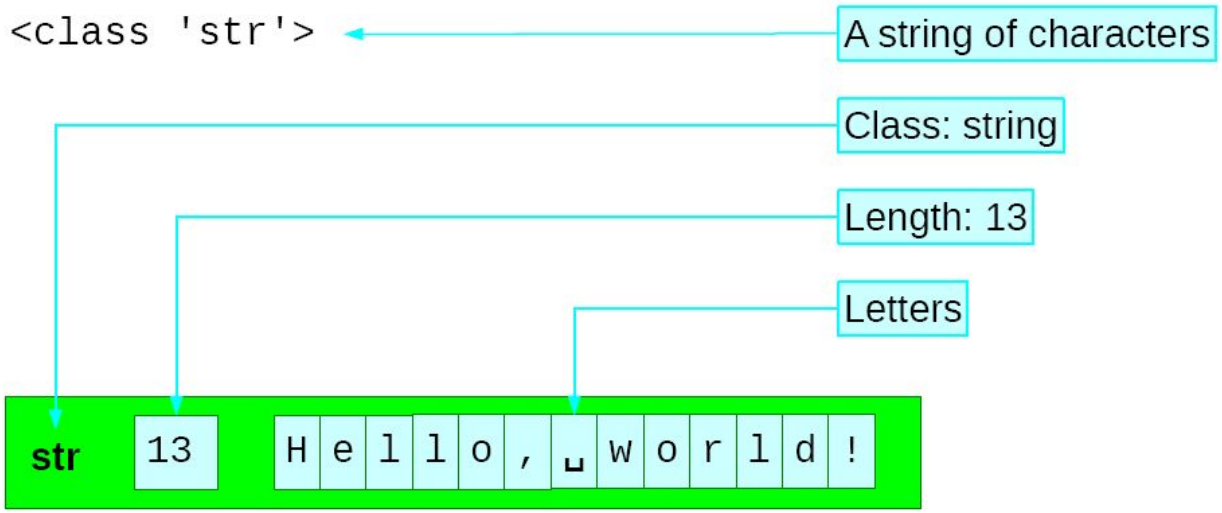
# Course outline (may change a bit)

- Part 2 Algorithm design and Programming in Python (balance 11 weeks):
  - Arithmetic/Logical/Boolean expressions and their evaluations in Python
  - Input/output statements (pseudo code, and in Python)
  - Assignment statement (pseudo code, and in Python)
  - Conditional statements, with sample applications
  - Iterative statements, with sample applications
  - Function sub-programs, arguments and scope of variables
  - Recursion
  - Modules
  - Specific data structures in Python (string, tuple, list, dictionary, array), with sample applications
  - Searching and sorting through arrays or lists
  - Handling exceptions
  - Classes, and object-oriented programming
  - (Time permitting) numerical methods: Newton Raphson, integration, vectors/matrices operations, continuous-time and discrete-event simulation

# String and string operations

- A string is a sequence of characters
- Strings are of type ‘str’
- Strings are enclosed in single quotes or double quotes

```
>>> type('Hello, world!')

<class 'str'>
```



# ASCII characters

- Character encoding is necessary to be able to represent them in binary form
- Two popular encoding schemes: ASCII and Unicode
- 8-bit ASCII:
  - It can represent 128 characters:
    - 96 printable characters including English/Latin letters, punctuation marks
      - i.e. **`a`**, …, **`z`**, **`A`**, …, **`Z`**, **`#`**, **`%`**, **`@`**, etc.
    - 32 control characters (such as **`SOH`**, **`STX`**, **`ETX`**)
  - the 8-th bit is the parity check
- Unicode:
  - Supports more than 120,000 different characters
  - UTF-8, UTF-16, UTF-32 are some of the Unicode encoding schemes
  - UTF-8 and ASCII are fully aligned

- Python by default uses UTF-8

# International characters using UTF-16, UTF-32

- Standard for encoding text expressed in most of the world's scripts
- Covering 154 modern and historic scripts
- 143,859 characters
- UTF-16: Uses '\u' followed by the hexadecimal (base 16) code for character
- Examples:

```
>>> print('\u011f')
ğ
>>> '\u0915'
क
>>> '\u0950'
ॐ
>>> '\u0967'
१
```

Read
Unicode 16 for Devanagari scri
About Unicode organization

ASCII and Unicode

Python 3.6
known limitations

```
1  print('\u011f')
2
3  print('\u0915')
4
5  print('\u0950')
6
7  print('\u0967')
```

Edit this code

Print outp

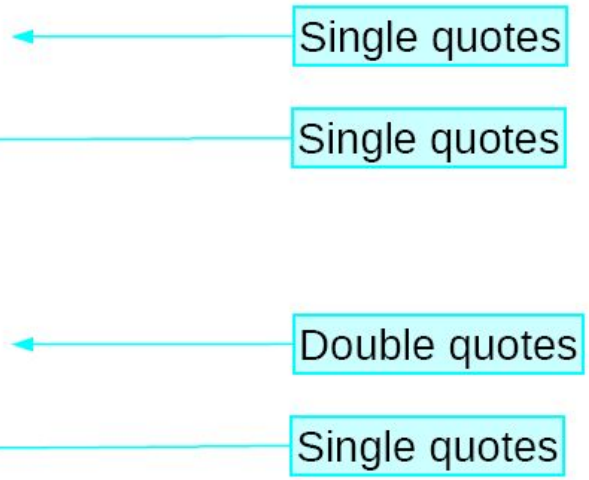
ğ
क
ॐ
१

Frames

# String and string operations

- Strings are enclosed in single quotes or double quotes

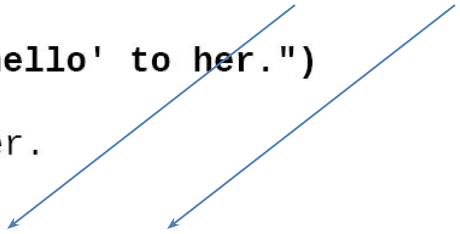
```
>>> 'Hello, world!'
'Hello, world!'

>>> "Hello, world!"
'Hello, world!'
```



# String and string operations

- String that contains single quotes or double quotes

```
>>> print('He said "hello" to her.')
He said "hello" to her.

>>> print("He said 'hello' to her.")
He said 'hello' to her.

>>> print('He said \'hello\' to her.')
He said 'hello' to her.
```



\' ⟶ '  Just an ordinary character.

\" ⟶ "  "Escaping"

# String and string operations

- Inserting special characters

```
>>> print('Hello,\nworld!')
Hello,
world!
```

# String and string operations

- Operators '**+**' and '*****'
  - `'Hello' + ' ' + 'World!'` → `'Hello World!'` ('**+**' is for concatenation)
  - `'John'*2` → 'JohnJohn' (* is for repetition)
  - Try out `2*'John'`, and see what happens
    - Useful to draw a line `10*'-'` will give `'----------'`

# String and string operations

- Operators '**+**' and '*****'
  - **'Hello' + ' ' + 'World!'** → **'Hello World!'** ('**+**' is for concatenation)
  - **'John'*2** → 'JohnJohn' (* is for repetition)
  - Try out **2*'John'**, and see what happens
    - Useful to draw a line **10*'-'** will give **'----------'**

# String and string operations

- Length of string:
  - **`len(s)`**

  Example:
  **`len('Hello')`** is 5, indexed from 0 through 4

```
>>> len('Hello,\nworld!')
13
```

`len()` function: gives the length of the object

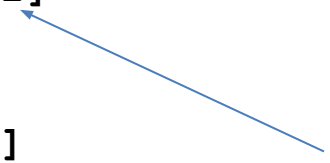
# String and string operations

- Indexing
  - An “index” is used to refer to and access individual character
  - Example:
    - ‘John’[0]
    - ‘John’[3]
    - ‘John’[4]
    - ‘John’[-1]
    - ‘John’[:]
    - ‘John’[:2]

# String, and string operations

- Indexing
  - An “index” is used to refer to and access individual or many characters in a string
  - Examples:

```
>>>'John'[0]
J

>>>'John'[3]
n

>>>'John'[4]
-- IndexError: string index out of range since len('John') is 4

>>>'John'[-1]
n

>>>'John'[:]
John

>>>'John'[:2]
Joh Jo
```



Read this as ‘what comes before 0’, viz.
`len('John') -1 = 3`

# String, and string operations

- Slicing a string == extracting a substring
- General syntax is

  **`s[start:end:step]`**

  where

  **`start`**: index to start slicing the string

  **`end`**: string is sliced until end-1

  **`step`**: determines the increment/decrement between each index for slicing

  Examples:

```
>>> s1 = "Hello World"
>>> print(s1[4:11:2])
oWrd

>>> s2 = "Hello"
>>> print(s2[1:len(s2):1]) # same as print(s2[1:5:1])
ello

>>> s3 = "Hello Howdee?"
>>> print(s3[0:-1:1])
Hello Howdee

>>> print(s3[-1])
?
```

Read this as 'what comes before 0', viz.

```
len(s3) -1 = 3
```

# Conversion between data types

`float()` Converts to floating point numbers

```
<class 'float'>
```

`int()` Converts to integers

```
<class 'int'>
```

`str()` Converts to strings

```
<class 'str'>
```

**`bool()`** Converts to booleans

```
<class 'bool'>
```

`''` → `False` Empty string

`'Fred'` → `True` Non-empty string

`0` → `False` Zero

`1` → `True` Non-zero

`12` → `True`

# Conversion between data types

```
# Conversion from xxx to float
print(float(2341))
print(float('20'))

#Conversion from xxx to int
print(int(2341.99))
print(int('20'))

# Conversion from xxx to str
print(str(2341))
print(str(2341.0))

# Conversion from xxx to bool
print(bool(''))
print(bool('Hari'))
print(bool(0))
print(bool(19))
```

Python 3.6
known limitations

```
1  # Conversion from xxx to float
2  print(float(2341))
3  print(float('20'))
4
5  #Conversion from xxx to int
6  print(int(2341.99))
7  print(int('20'))
8
9  # Conversion from xxx to str
10 print(str(2341))
11 print(str(2341.0))
12
13 # Conversion from xxx to bool
14 print(bool(''))
15 print(bool('Hari'))
16 print(bool(0))
17 print(bool(19))
```

Print outp

```
2341.0
20.0
2341
20
2341
2341.0
False
True
False
True
```

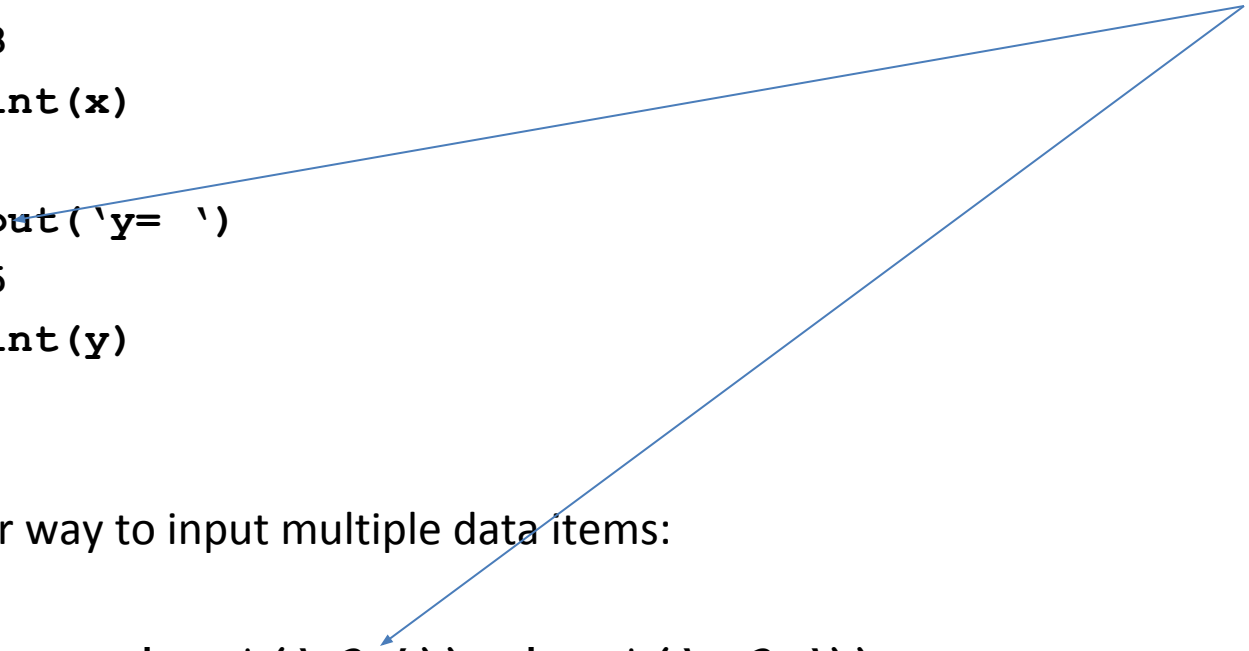
Frames

See also:
https://tinyurl.com/yukwsf22

# Input multiple data items

One way to input no. of data items:

```
>>>input('x= ')
x= 123
>>>print(x)
123
>>>Input('y= ')
y= 345
>>>print(y)
345
```



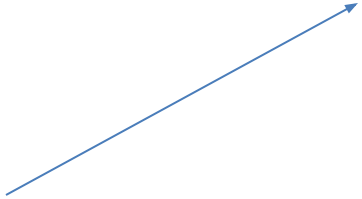
Another way to input multiple data items:

```
>>> x, y = input('x? '), input(' y? '))
x? 123 y? 345
>>> print('x = ', float(x), 'y = ', float(y))
x = 123.0 y = 345.0
```

# Input multiple data items

Yet another way to input no. of data items

```
# taking multiple inputs at a time
x, y, z = input('Enter no. of books in English, Hindi, Urdu:').split()
print('Number of books in English: ', int(x))
print('Number of books in Hindi: ', int(y))
print('Number of books in Urdu: ', int(z))
```



split() method to split a Python string using a “separator” (e.g. “space”)

# Input multiple data items

Yet another way to input no. of data items

```
# taking multiple inputs at a time
x, y, z = input('Enter no. of books in English, Hindi, Urdu:').split()
print('Number of books in English: ', int(x))
print('Number of books in Hindi: ', int(y))
print('Number of books in Urdu: ', int(z))
```

split() method to split a Python string using a “separator” (e.g. “space”)

Python 3.6
known limitations

```
1 x, y, z = input('Enter no. of books in English, Hindi, U
2
3 print('Number of books in English: ', int(x))
4
5 print('Number of books in Hindi: ', int(y))
6
7 print('Number of books in Urdu: ', int(z))
```

Edit this code

line that just executed

Print output (drag lower right corner to resize)

```
Enter no. of books in English, Hindi, Urdu: 22
Number of books in English:  22
Number of books in Hindi:  11
Number of books in Urdu:  5
```

Frames Objects

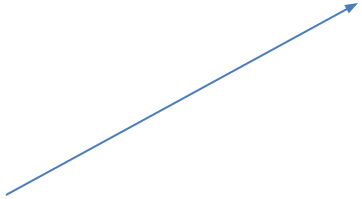
Global frame

| | |
|---|---|
| x | "22" |
| y | "11" |
| z | "5" |

# Input multiple data items

Yet another way to input no. of data items

```
# taking multiple inputs at a time
x, y, z = input('Enter no. of books in English, Hindi, Urdu:').split()
print('Number of books in English: ', int(x))
print('Number of books in Hindi: ', int(y))
print('Number of books in Urdu: ', int(z))
```



split() method to split a Python string using a “separator” (e.g. “space”)

What if the strings are separated by ‘ : ’? As in … **`split(:)`**

# Conditional statements

- Pseudo code (or algorithmic statement):

```
if C1 then S1
```

- In Python:

```
if C1:
    S1
```

- Example:

```
INC = float(input('Your Income? '))
Tax = 0
if INC > 100000:
    Tax = 0.1*(INC-100000)
print('Income is ', INC, 'Tax is ', Tax)
```

# Conditional statements

- Pseudo code (or algorithmic statement):

  **if C1 then S1**

- In Python:

```
if C1:
    S1
```

- Example:

```
INC = float(input('Your Income? '))
Tax = 0
if INC > 100000:
    Tax = 0.1*(INC-100000)
print('Income is ', INC, 'Tax is ', Tax)
```

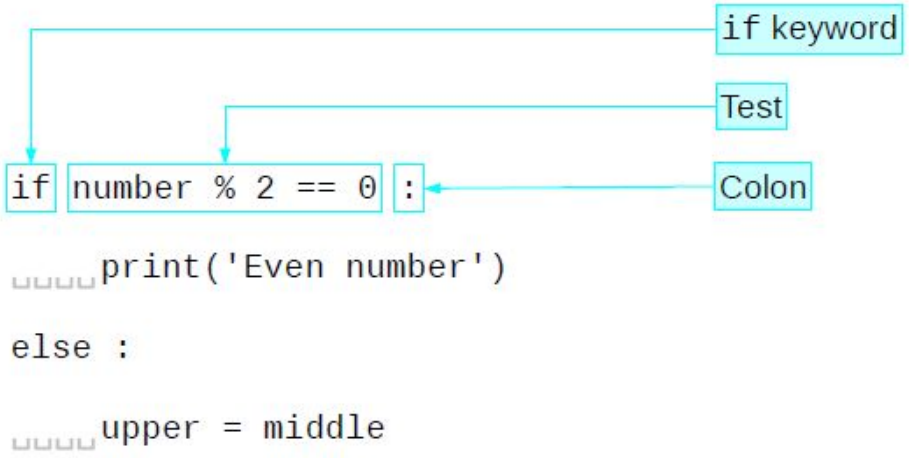
# Conditional statements

- Pseudo code:

```
if C1 then S1 else S2
```

- In Python:

```
if C1:
    S1
else:
    S2
```




- Example:

```
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
print('T1, T2 ', T1, T2)
if(T1 < T2):
    minT = T1
else:
    minT = T2
print(T1, T2, minT)
```

# Conditional statements

- Pseudo code:

```
if C1 then S1 else S2
```

- In Python:

```
if C1:
    S1
else:
    S2
```

- Example:

```
T1 = float(input('Time 1? '))
T2 = float(input('Time 2? '))
print('T1, T2 ', T1, T2)
if(T1 < T2):
    minT = T1
else:
    minT = T2
print(T1, T2, minT)
```

# Conditional statements

- Pseudo code:

```
if C1 then S1 else [if C2 then S2]
```

- In Python:

```
if C1:
    S1
elif C2:
    S2
```

Read it as 'else if'

- Example:

```
INC = float(input('Your Income? '))
Tax = 0
if INC > 200000:
 Tax = 10000 + 0.2*(INC-200000)
elif INC > 100000:
 Tax = 0.1*(INC-100000)
print('Income is ', INC, 'Tax is ', Tax)
```

# Conditional statements

- Pseudo code:

```
if C1 then S1 else [if C2 then S2]
```

- In Python:

```
if C1:
    S1
elif C2:
    S2
```

- Example:

```
if x%2 == 0:
    if x%3 == 0:
        print(x, 'is divisible by 2 and 3')
    else:
        print(x, 'is divisible by 2 but not by 3')
elif x%3 == 0:
    print(x, 'is divisible by 3 but not by 2')
```

# Iteration

Python supports while and for loops

- Pseudo code
  while C do S

- In Python
  while C:
  ␣␣␣␣S

# Iteration

Python supports while and for loops

- Pseudo code:

```
# Find the largest n such that 2**n ≤ 50
n = 0; x = 2**n;
while x ≤ 50 do [n = n+1; x = 2**n];
output('largest n such that 2**n ≤ 50 is ', n-1)
```

- In Python:

```
# Find the largest n such that 2^n ≤ 50
n = 0
x = 2**n
while x <= 50:
    n = n+1
    x = 2**n
print('largest n such that 2^n <= 50 is ', n-1)
```

- Question: what will be the output?

| Test of condition | n | x | x<=50 |
|---|---|---|---|
| 1 st | 0 | 1 | TRUE |
| 2 nd | 1 | 2 | TRUE |
| 3 rd | 2 | 4 | TRUE |
| 4 th | 3 | 8 | TRUE |
| 5 th | 4 | 16 | TRUE |
| 6 th | 5 | 32 | TRUE |
| 7 th | 6 | 64 | FALSE |

# Iteration

- Example: computing square root y = √x, where x > 0

  Somewhat informal version of an algorithm

  1. **Start with a guess, g = x/2 # for instance**
  2. **if |g*g – x| is small**

     **then [conclude g = √x; output(g); stop]**

     **else [compute new guess g = (g + x/g)/2; repeat step 2]**

**Example outcomes:**

**Let x = 3**

| Round | g | \|g*g-x\| |
|---|---|---|
| 1 | 1.5 | 0.75 |
| 2 | 1.75 | 0.0625 |
| 3 | 1.732143 | 0.000319 |

**or x = 16**

| Round | g | \|g*g-x\| |
|---|---|---|
| 1 | 8 | 48 |
| 2 | 5 | 9 |
| 3 | 4.1 | 0.81 |
| 4 | 4.00122 | 0.009758 |

# Iteration

**Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java**

Python 3.6
known limitations

```
1  # compute sqrt(x), x > 1
2  x, epsilon = float(input()), 0.0001
3  print( 'x = ', x, 'epsilon = ', epsilon)
4  g = x/2                    #initial guess
5  while abs(g*g - x) >= epsilon:
6      g = (g + x/g)/2        # new guess
7      print('g = ', g)
8  print('SQRT(x) is ', g, 'given abs(g*g - x) < epsilon')
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (17 steps)

NEW: if you use ChatGPT or AI, take this survey

Move and hide objects

Print output (drag lower right corner to resize)

```
16
x =  16.0 epsilon =  0.0001
g =  5.0
g =  4.1
g =  4.001219512195122
g =  4.0000001858445895
SQRT(x) is  4.0000001858445895 given abs(g*g - x)
```

Frames Objects

Global frame

| | |
|---|---|
| x | 16.0 |
| epsilon | 0.0001 |
| g | 4 |

# Iteration

- Example: computing square root g = √x, where x > 1

  Another algorithm, based on "bisection method"

Python 3.6
known limitations

```
1  # compute sqrt(x), x > 1
2  x, epsilon = float(input()), 0.0001
3  print('x = ', x, 'epsilon = ', epsilon)
4  low, high = 0, x
5  g = (low+high)/2          #initial guess
6  print(g)
7  while abs(g*g - x) >= epsilon:
8      if g*g < x:
9          low = g          # no change in high
10     else:
11         high = g          # no change in low
12     g = (low+high)/2      # new better guess
13     print(g)
14 print('SQRT(x) is ', g, 'given abs(g*g - x) < epsilon')
```

Edit this code

ine that just executed
ext line to execute

Print output (drag lower right corner to resize)

```
16
x =  16.0 epsilon =  0.0001
8.0
4.0
SQRT(x) is  4.0 given abs(g*g - x) < epsilon
```

Frames Objects

Global frame

| | |
|---|---|
| x | 16.0 |
| epsilon | 0.0001 |
| low | 0 |
| high | 8.0 |
| g | 4.0 |

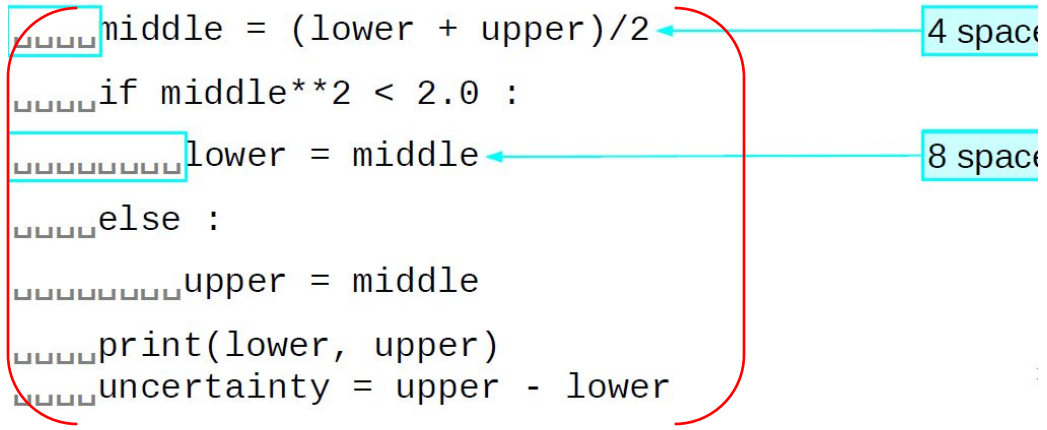
# A note on indentation

Beware: In Python indentation matters:

In pseudo code:

```
# compute the SQRT of 2.0
tolerance = 1.0 e-15;
lower = 0.0;
upper = 2.0;
uncertainty = upper-lower;
while uncertainty > tolerance do
    [middle = (lower + upper)/2;
    if middle**2 < 2.0
    then lower = middle
    else upper = middle;
    print(lower, upper);
    uncertainty = upper-lower
    ]
```

```
tolerance = 1.0e-15
lower = 0.0
upper = 2.0
uncertainty = upper - lower

while uncertainty > tolerance :
    middle = (lower + upper)/2
    if middle**2 < 2.0 :
        lower = middle
    else :
        upper = middle
    print(lower, upper)
    uncertainty = upper - lower
```

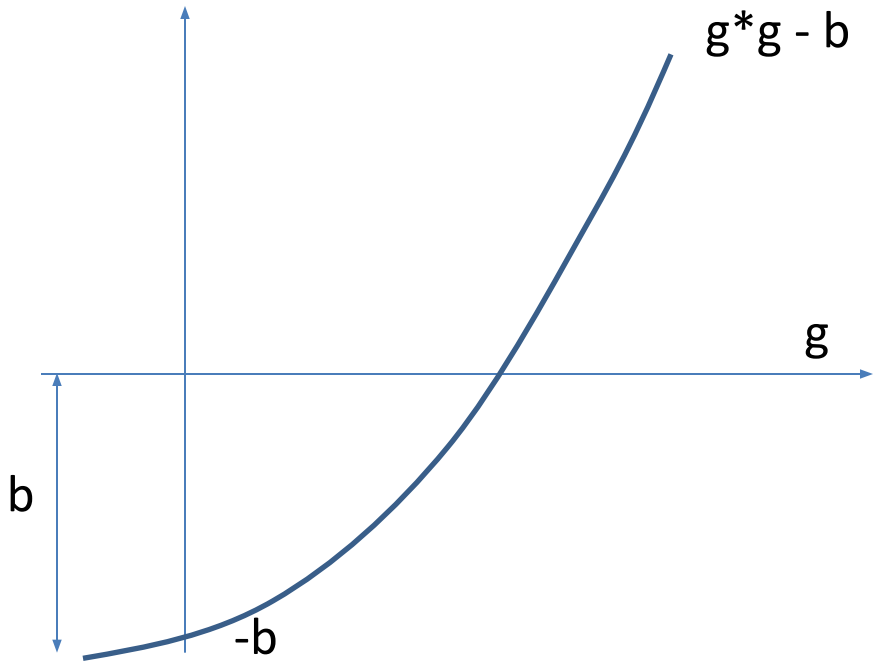


# Iteration

- Example: computing square root x = √k, k > 0, or solving equation x^2 - k = 0

  Another algorithm, based on "Newton-Raphson method" where we try solve for g such that equation g*g - b = 0

```
# compute sqrt(b), where b > 0
b = float(input()), epsilon = 0.0001
g = b/2                  #initial guess
while abs(g*g - b) >= epsilon:
    g = g - ((g*g - b)/(2*g))  #new better guess
    print(g)
print(g)
```



# Iteration

- Example: computing square root x = √k, k > 0, or solving equation x^2 - k = 0

  Another algorithm, based on "Newton-Raphson method" where we try solve for g such that equation g*g - b = 0

..

Python 3.6
known limitations

```
1  # compute sqrt(x), x > 1 using Newton-Raphson method
2  x, epsilon = float(input()), 0.0001
3  print('x = ', x, 'epsilon = ', epsilon)
4  g = x/2              #initial guess
5  print('Initial guess ', g)
6  while abs(g*g - x) >= epsilon:
7      g = g - ((g*g-x)/(2*g))      # new better guess
8      print('New guess ', g)
9  print('SQRT(x) is ', g, 'given abs(g*g - x) < epsilon')
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (18 steps)

Print output (drag lower right corner to resize)

```
16
x =  16.0 epsilon =  0.0001
Initial guess  8.0
New guess  5.0
New guess  4.1
New guess  4.001219512195122
New guess  4.0000001858445895
SQRT(x) is  4.0000001858445895 given abs(g*g
```

Frames Objects

Global frame

| | |
|---|---|
| x | 16.0 |
| epsilon | 0.0001 |
| g | 4 |

# Iteration – break command

- **`break`** commend
  - Used to terminate the loop when **`break`** statement is encountered
  - Improves efficiency (need not wait until loop terminates)
  - Control is transferred to statement following loop

- Example (in Python):

```
#Find the smallest +ve integer divisible by 11 & by 12
x = 1
while True:
    if x%11 == 0 and x%12 == 0:
        break
    x = x + 1
print(x, "is divisible by 11 and 12")
```

Output:

```
132 is divisible by 11 and 12
```

# Iteration using 'for' statement

Python supports **`for`** loops

- Pseudo code

  **`for k in <sequence> do S`**

  where **`<sequence>`** is an ordered set of objects, typically integers, strings, etc.

- In Python:

```
for k in [sequence]:
 S
```

- Example:

```
pets = ['cat','dog','cow']
for k in pets:
    print(k)
```

- Another example:

```
for k in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]:
    if (k%2 == 0 and k%3 == 0):
        print(k, 'is divisible by 2 and 3')
print('That is it')
```

# Iteration

Python supports **`for`** loops

- Pseudo code

  **`for k in <sequence> do S`**

  where **`<sequence>`** is an ordered set of objects, typically integers, strings, etc.

- In Python:

```
for k in [sequence]:
 S
```

- Example:

```
pets = ['cat','dog','cow']
for k in pets:
    print(k)
```

- Another example:

```
for k in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]:
    if (k%2 == 0 and k%3 == 0):
        print(k, 'is divisible by 2 and 3')
print('That is it')
```

# Iteration

Python supports **`for`** loops

- Pseudo code

  **`for k in <sequence> do S`**

  where **`<sequence>`** is an ordered set of objects, typically integers, strings, etc.

- In Python:

```
for k in [sequence]:
 S
```

- Example:

```
pets = ['cat','dog','cow']
for k in pets:
    print(k)
```

- Another example:

```
for k in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]:
    if (k%2 == 0 and k%3 == 0):
        print(k, 'is divisible by 2 and 3')
print('That is it')
```

# Iteration

- Using the **range(., ., .)** function instead of:

```
for k in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]:
    if (k%2 == 0 and k%3 == 0):
        print(k, 'is divisible by 2 and 3')
print('That is it')
```

- Another way to write the above:

```
for k in range(1,19):
    if (k%2 == 0 and k%3 == 0):
        print(k, 'is divisible by 2 and 3')
print('that is it')
```

```
for k in range(1,19):
```

Is effectively the same as:

```
for k in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]:
```

# The `range` function

- A built-in function that generates an ordered sequence of numbers
- It has three arguments: start, stop and step

`range(start, stop, step)`
  - `start` is optional, and defaults to `0`
  - `step` is optional, and defaults to `1`

- Example program segment:

```
# print numbers from 0 to 9, both inclusive
for i in range(10):
    print(i)
```

- Examples of `range` calculation
  - `range(6)` generates `[0, 1, …, 5]`
  - `range(2,6)` generates `[2, 3, 4, 5]`
  - `range(0,6,2)` generates `[0, 2, 4]`
  - `range(6,0,-2)` generates `[6, 4, 2]`

# The `range` function

- A built-in function that generates an ordered sequence of numbers
- It has three arguments: start, stop and step

`range(start, stop, step)`
  - `start` is optional, and defaults to `0`
  - `step` is optional, and defaults to `1`

- Example program segment:

```
# print numbers from 0 to 9, both inclusive
for i in range(10):
    print(i)
```

`range(10) is == range(0, 10, 1)`

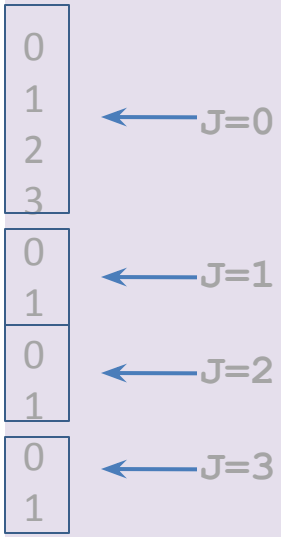
- Examples of `range` calculation
  - `range(6) == range(0, 6, 1)` -- generates `[0, 1, …, 5]`
  - `range(2,6) == range(2, 6, 1)` -- generates `[2, 3, 4, 5]`
  - `range(0,6,2)` generates `[0, 2, 4]`
  - `range(6,0,-2)` generates `[6, 4, 2]`

# The `range` function

- Another interesting example to demo evaluation of range(start, stop, step) :

```
x = 4
for j in range(x):
    for i in range(x):
        print(i)
        x = 2
```

Output is:




Why?

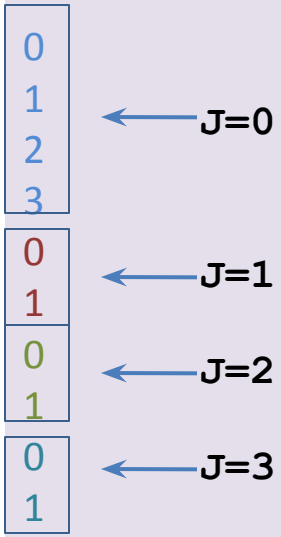
The arguments of range function are evaluated just before the first iteration and not reevaluated for every iteration

# The `range` function

- Another interesting example to demo evaluation of range(start, stop, step) :

```
x = 4
for j in range(x):
    for i in range(x):
        print(i)
        x = 2
```

Output is:




Why?

The arguments of range function are evaluated just before the first iteration and not reevaluated for every iteration

# Q&A

- On strings, and characters (ASCII, UTF-16)
- On input statement, using split
- On conditional statements in Python
- On iteration using while statement
- On iteration using for statement
- On break function
- On range function

- https://docs.google.com/forms/d/e/1FAIpQLSdcRv7VzRSjK2sK5UuJ8k8q5zLCwRjRA844xIYYJUWElNkAyg/viewform?usp=sf_link