A Quick Tour of Python

Amey Karkare
Dept. of CSE
IIT Kanpur

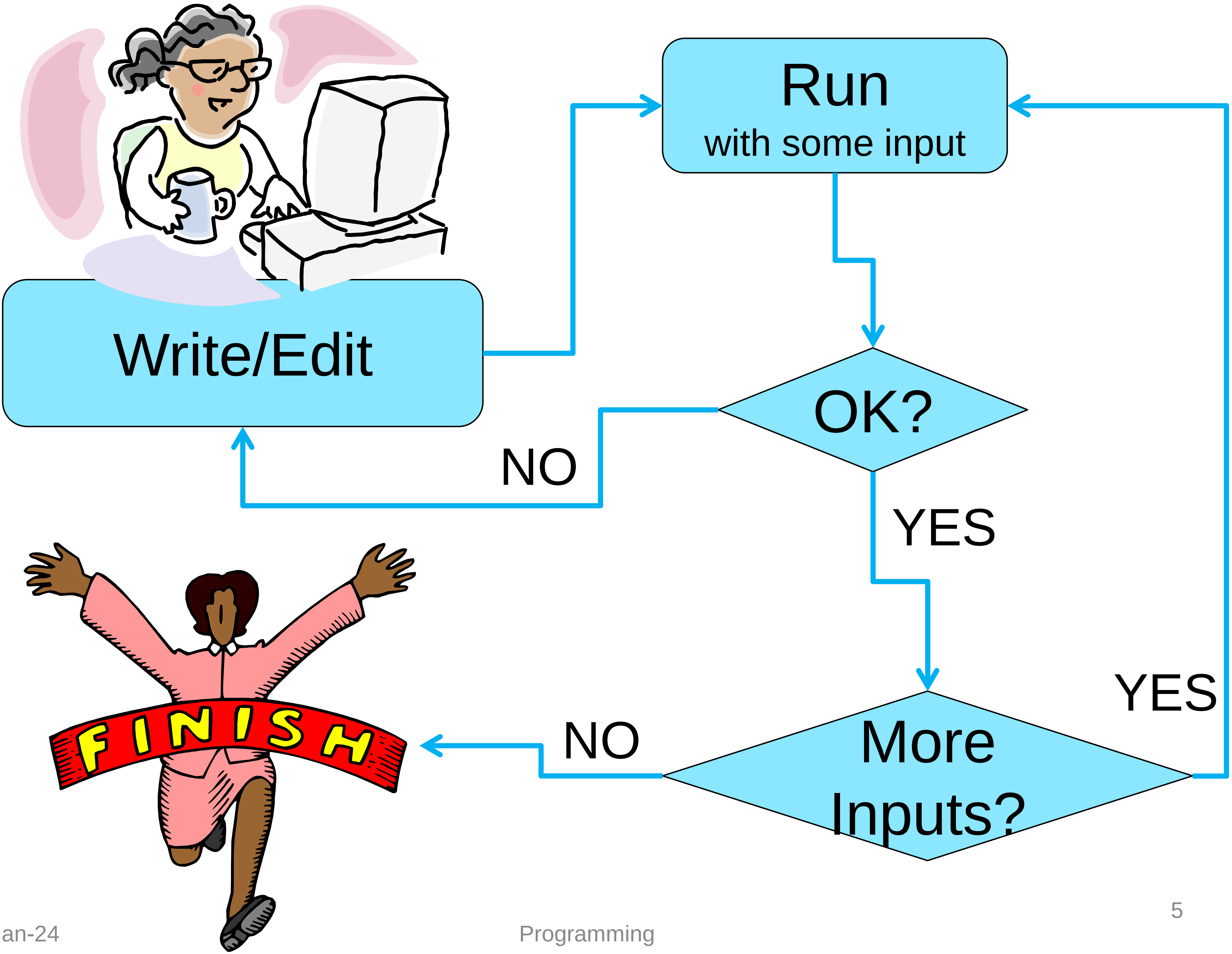
Acknowledgements

- MS Office clip art, various websites and images
  - The images/contents are used for teaching purpose and for fun. The copyright remains with the original creator. If you suspect a copyright violation, bring it to my notice and I will remove that image/content.

About me

http://www.cse.iitk.ac.in/~karkare

The Programming Cycle for Python



Run

with some input

Write/Edit

OK?

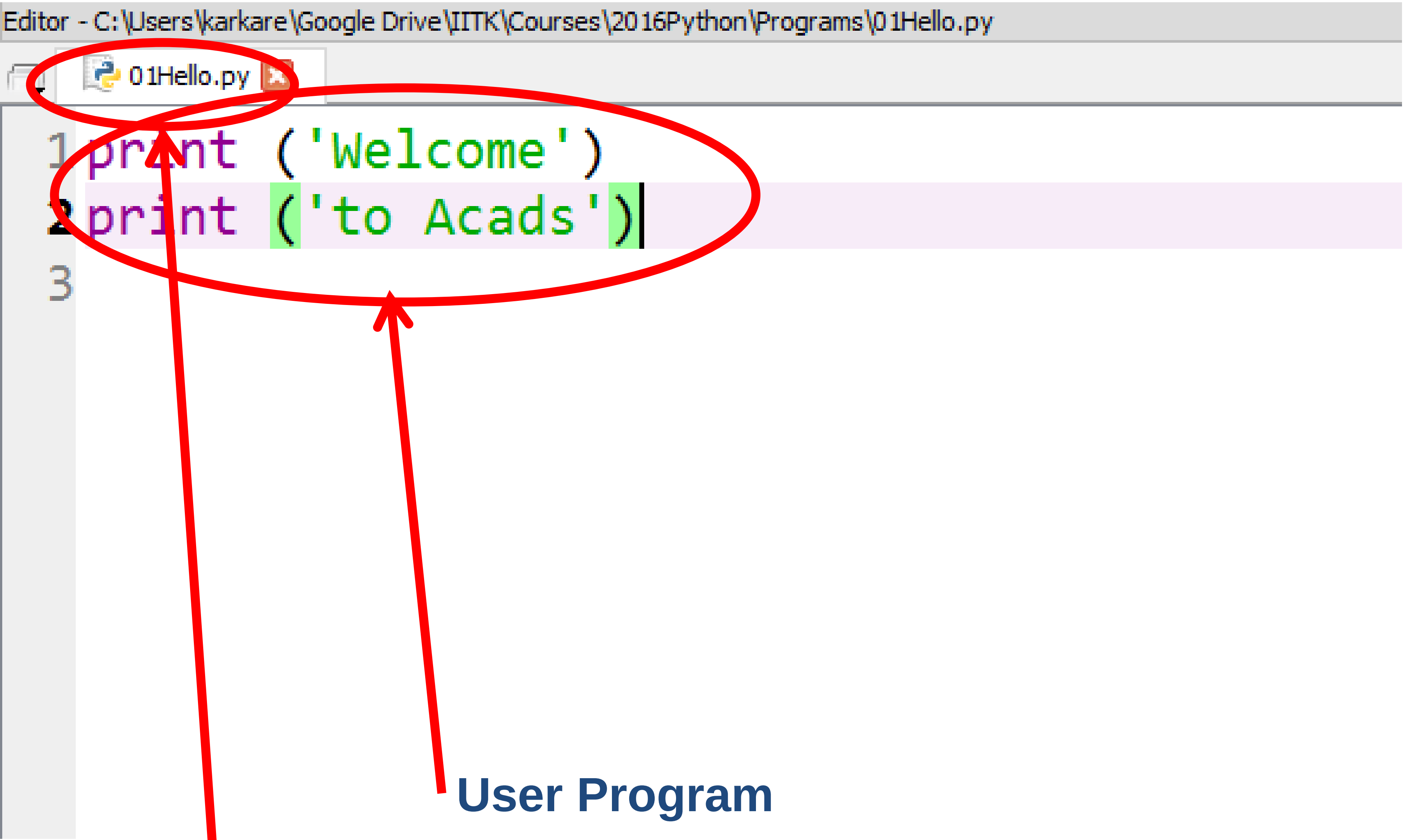
NO

YES

More Inputs?

YES

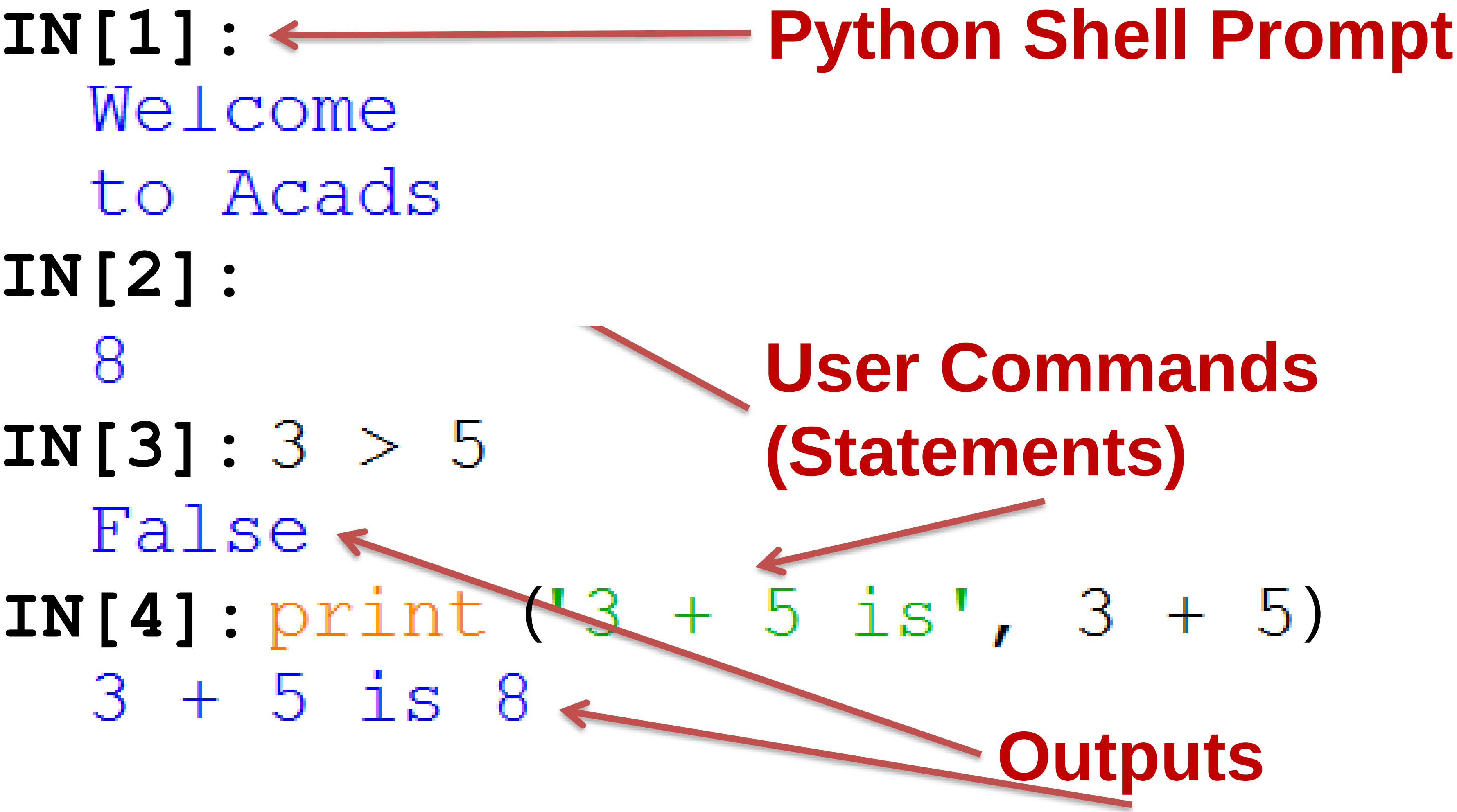
NO


Editor - C:\Users\karkare\Google Drive\IITK\Courses\2016Python\Programs\01Hello.py
01Hello.py

```
print ('Welcome')
print ('to Acads')
```

User Program

Filename, preferred extension is py



```
IN[1]:
   Welcome
   to Acads
IN[2]:
   8
IN[3]: 3 > 5
   False
IN[4]: print ('3 + 5 is', 3 + 5)
   3 + 5 is 8
```

Python Shell Prompt

User Commands (Statements)

Outputs

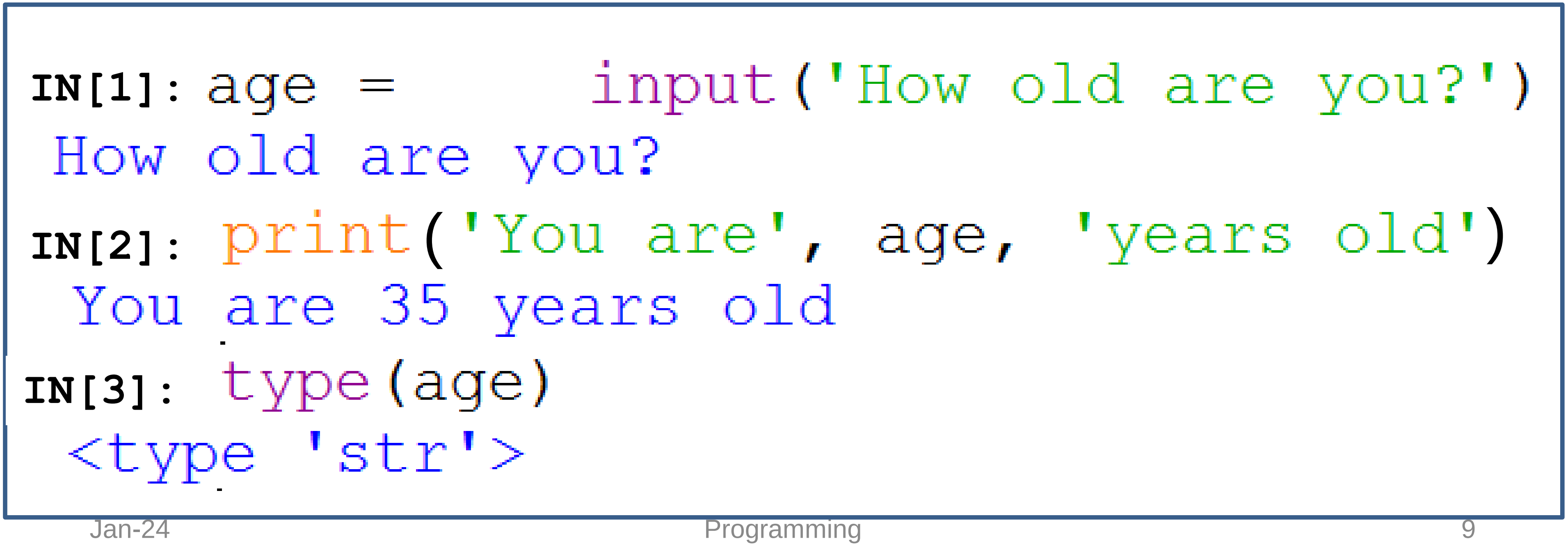
Python Shell is Interactive

Interacting with Python Programs

- Python program communicates its results to user using print
- Most useful programs require information from users
  - Name and age for a travel reservation system
- Python 3 uses input to read user input as a string (str)

input

- Take as argument a string to print as a prompt
- Returns the user typed value as a string
  - details of how to process user string later



```
IN[1]: age =     input('How old are you?')
 How old are you?
IN[2]: print('You are', age, 'years old')
 You are 35 years old
IN[3]: type(age)
 <type 'str'>
```

Elements of Python

- A Python program is a sequence of **definitions** and **commands (statements)**
- Commands manipulate **objects**
- Each object is associated with a **Type**
- **Type:**
  - A set of values
  - A set of operations on these values
- **Expressions**: An operation (combination of objects and **operators**)

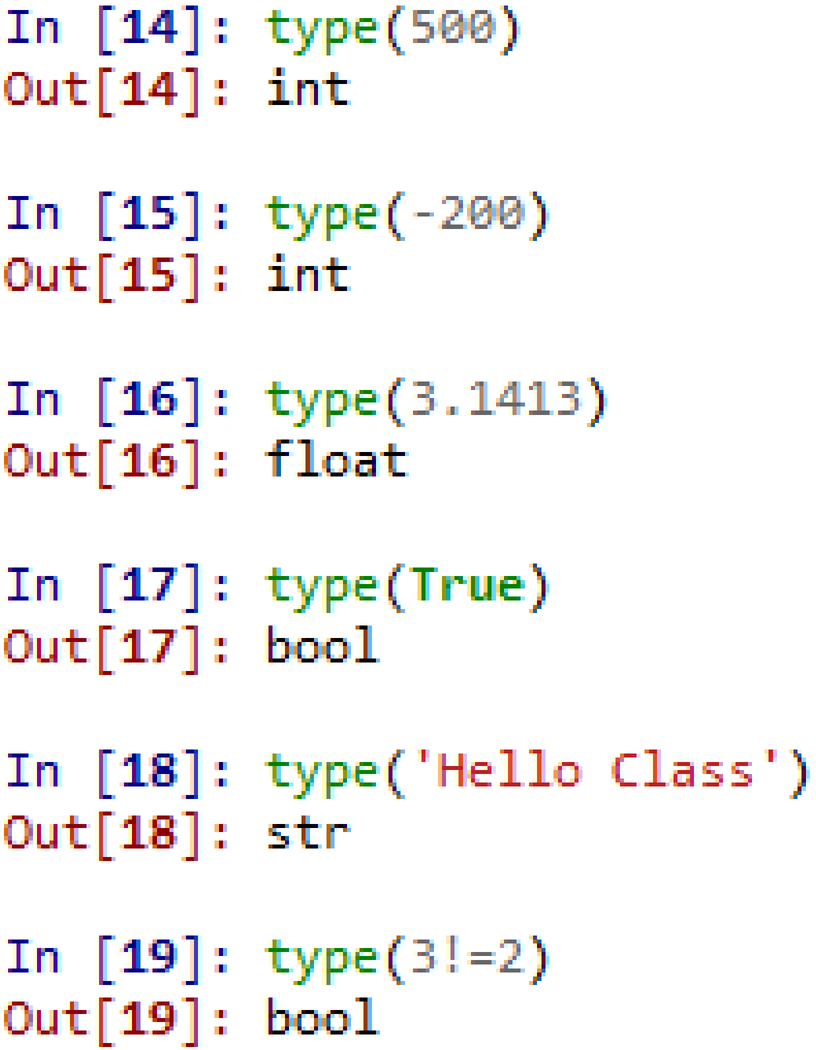
Types in Python

- **int**
  - Bounded integers, e.g. 732 or -5
- **float**
  - Real numbers, e.g. 3.14 or 2.0
- **long**
  - Long integers with unlimited precision
- **str**
  - Strings, e.g. ‘hello’ or ‘C’

Types in Python

- **Scalar**
  - Indivisible objects that do not have internal structure
  - **int** (signed integers), **float** (floating point), **bool** (Boolean), ***NoneType***
    - NoneType is a special type with a single value
    - The value is called **None**
- **Non-Scalar**
  - Objects having internal structure
  - **str** (strings)

Example of Types

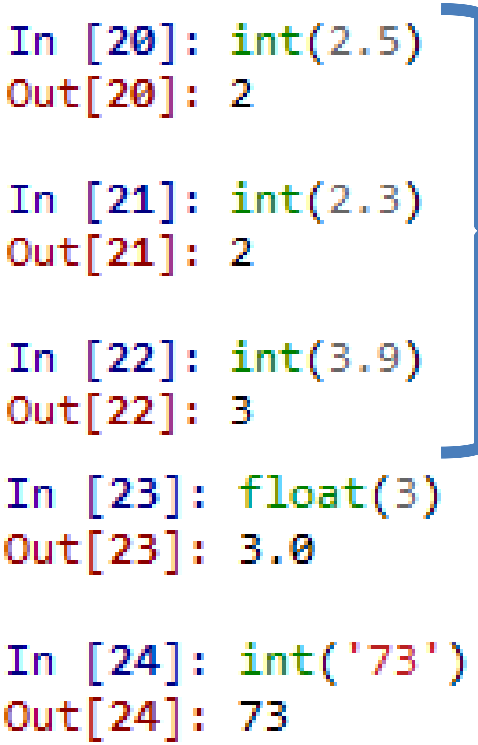


```
In [14]: type(500)
Out[14]: int

In [15]: type(-200)
Out[15]: int

In [16]: type(3.1413)
Out[16]: float

In [17]: type(True)
Out[17]: bool

In [18]: type('Hello Class')
Out[18]: str

In [19]: type(3!=2)
Out[19]: bool
```

Type Conversion (Type Cast)

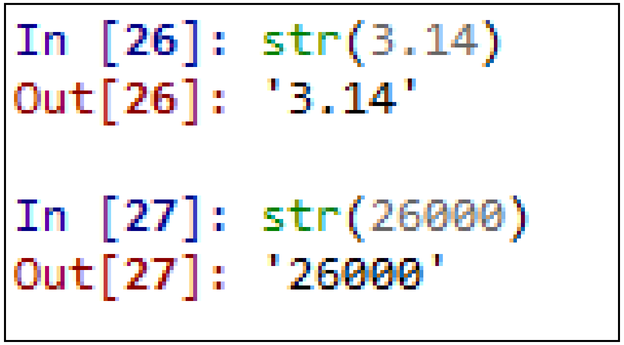
- Conversion of value of one type to other
- We are used to int ↔ float conversion in Math
  - Integer 3 is treated as float 3.0 when a real number is expected
  - Float 3.6 is truncated as 3, or rounded off as 4 for integer contexts
- Type names are used as type converter functions

Type Conversion Examples

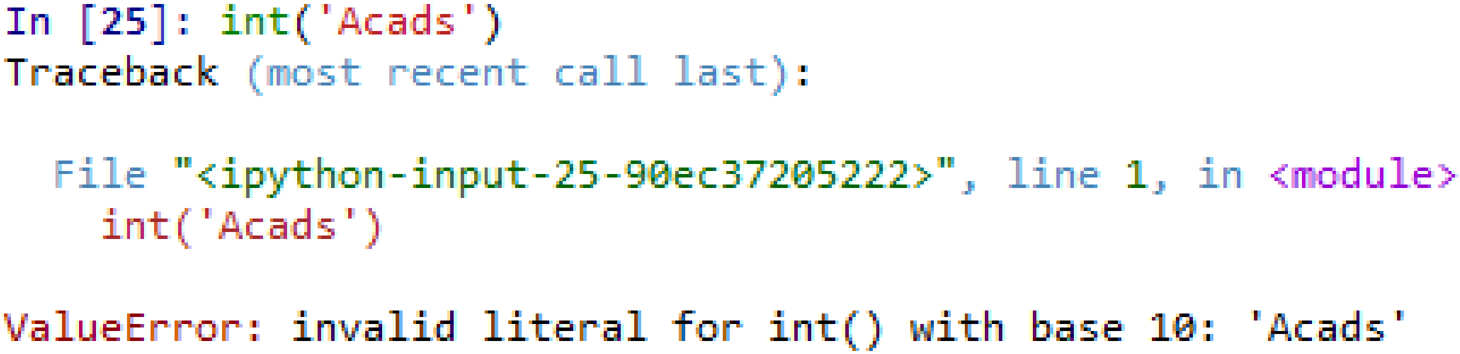


```
In [20]: int(2.5)
Out[20]: 2

In [21]: int(2.3)
Out[21]: 2

In [22]: int(3.9)
Out[22]: 3

In [23]: float(3)
Out[23]: 3.0

In [24]: int('73')
Out[24]: 73
```

Note that float to int conversion is truncation, not rounding off

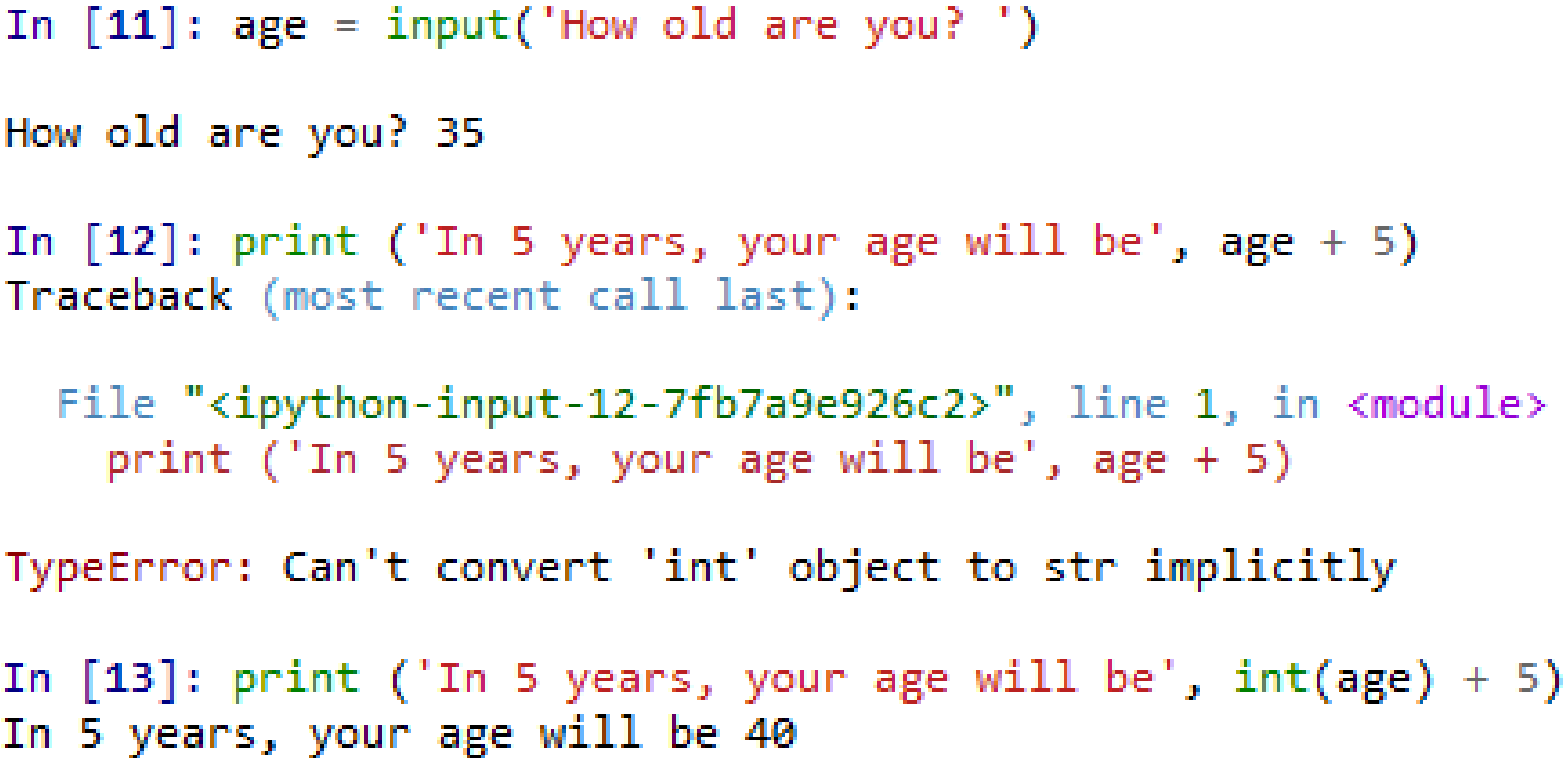


```
In [26]: str(3.14)
Out[26]: '3.14'

In [27]: str(26000)
Out[27]: '26000'
```



```
In [25]: int('Acads')
Traceback (most recent call last):

  File "<ipython-input-25-90ec37205222>", line 1, in <module>
    int('Acads')

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

Type Conversion and Input



```
In [11]: age = input('How old are you? ')

How old are you? 35

In [12]: print ('In 5 years, your age will be', age + 5)
Traceback (most recent call last):

  File "<ipython-input-12-7fb7a9e926c2>", line 1, in <module>
    print ('In 5 years, your age will be', age + 5)

TypeError: Can't convert 'int' object to str implicitly

In [13]: print ('In 5 years, your age will be', int(age) + 5)
In 5 years, your age will be 40
```

Operators

• Arithmetic	+	-	*	//	/	%	**	
• Comparison	==	!=	>	<	>=	<=		
• Assignment	=	+=	-=	*=	//=	/=	%=	**=
• Logical	and	or	not					
• Bitwise	&	|	^	~	>>	<<		
• Membership	in	not in						
• Identity	is	is not						

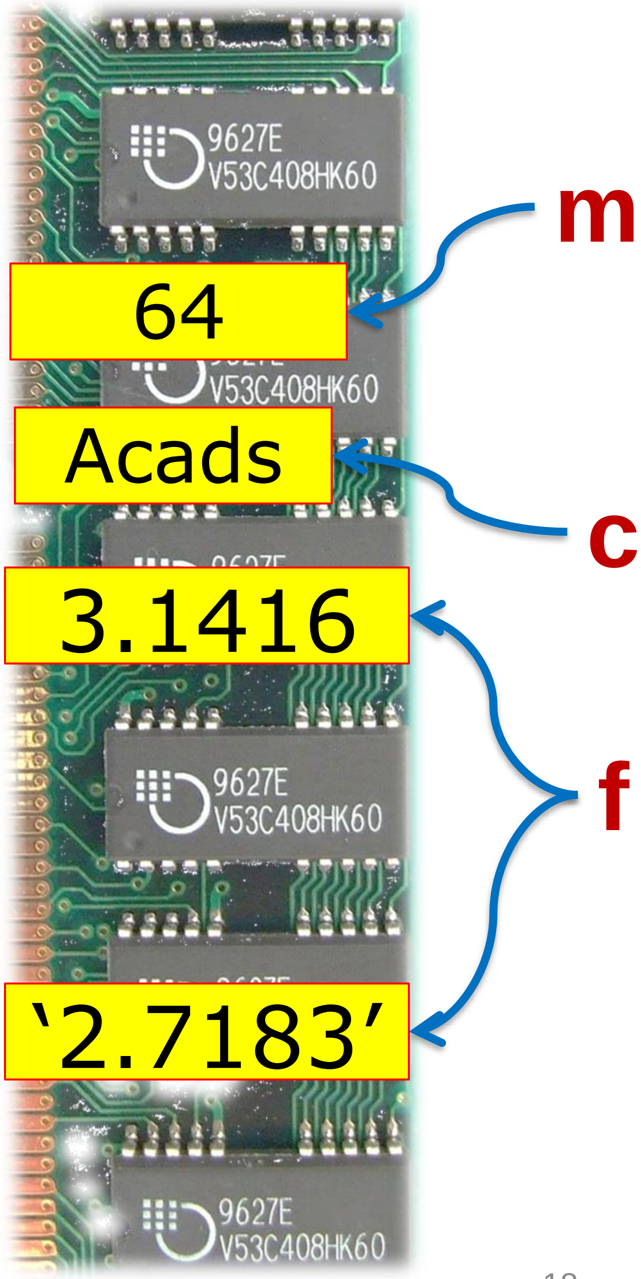
Variables

- A name associated with an object
- Assignment used for binding

```
m = 64;
c = ‘Acads’;
f = 3.1416;
```

- Variables can change their bindings

```
f = ‘2.7183’;
```



64

Acads

3.1416

‘2.7183’

m

c

f

Assignment Statement

- A simple assignment statement

  Variable = Expression

- Computes the value (object) of the expression on the right hand side expression (RHS)
- Associates the name (variable) on the left hand side (LHS) with the RHS value
- = is known as the assignment operator.

Multiple Assignments

- Python allows multiple assignments

```
x, y = 10, 20
```

Binds x to 10 and y to 20

- Evaluation of multiple assignment statement:
  - All the expressions on the RHS of the = are first evaluated **before any binding happens**.
  - Values of the expressions are bound to the corresponding variable on the LHS.

```
x, y = 10, 20
x, y = y+1, x+1
```

x is bound to 21 and y to 11 at the end of the program

Programming using Python

Operators and Expressions

Binary Operations

Op	Meaning	Example	Remarks
+	Addition	9+2 is 11	
		9.1+2.0 is 11.1	
-	Subtraction	9-2 is 7	
		9.1-2.0 is 7.1	
*	Multiplication	9*2 is 18	
		9.1*2.0 is 18.2	
/	Division	9/2 is 4.25	In Python3
		9.1/2.0 is 4.55	Real div.
//	Integer Division	9//2 is 4	
%	Remainder	9%2 is 1	

The // operator

- Also referred to as “integer division”
- Result is a whole integer (floor of real division)
  - But the type need not be int
  - the integral part of the real division
  - rounded towards minus infinity $(-\infty)$
- Examples

9//4 is 2	**(-1)//2 is -1**	**(-1)//(-2) is 0**
1//2 is 0	**1//(-2) is -1**	**9//4.5 is 2.0**

The % operator

- The remainder operator % returns the remainder of the result of dividing its first operand by its second.

9%4 is 1	**(-1)%2 is 1**	**(-1)//(-2) is 0**
9%4.5 is 0.0	**1%(-2) is 1**	**1%0.6 is 0.4**

Ideally: x == (x//y)*y + x %y

Conditional Statements

- In daily routine
  - If it is very hot, I will skip exercise.
  - If there is a quiz tomorrow, I will first study and then sleep. Otherwise I will sleep now.
  - If I have to buy coffee, I will go left. Else I will go straight.

if-else statement

- Compare two integers and print the min.

```
if  x < y:
        print (x)
else:
        print (y)
print (‘is the minimum’)
```

1. Check if x is less than y.
2. If so, print x
3. Otherwise, print y.

Indentation

- Indentation is **important** in Python
  - grouping of statement (block of statements)
  - no explicit brackets, e.g. { }, to group statements

```
x,y = 6,10

if x < y:
    print (x)
else:
     print (y)
     print (‘is t[illegible]n’)
```

skipped

Run the program

6

10

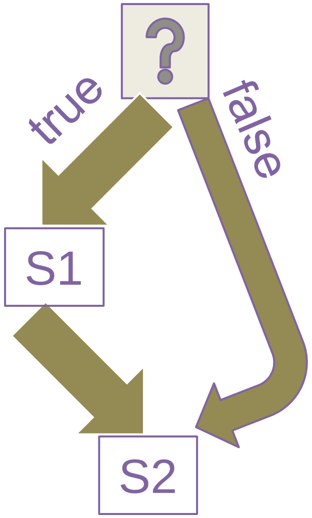
Output

6

if statement (no else!)

- General form of the if statement

```
if boolean-expr :
        S1
S2
```

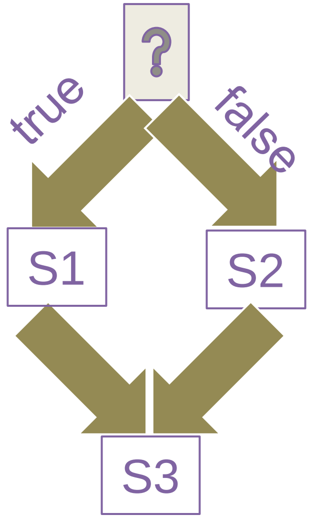


- Execution of if statement
  - First the expression is evaluated.
  - If it evaluates to a **true** value, then S1 is executed and then control moves to the S2.
  - If expression evaluates to **false**, then control moves to the S2 directly.

if-else statement

- General form of the if-else statement

```
if boolean-expr :
        S1
else:
        S2
S3
```



- Execution of if-else statement
  - First the expression is evaluated.
  - If it evaluates to a **true** value, then S1 is executed and then control moves to S3.
  - If expression evaluates to **false**, then S2 is executed and then control moves to S3.
  - S1/S2 can be **blocks** of statements!

Nested if, if-else

```
if a <= b:
        if a <= c:
            …
        else:
             …
else:
         if b <= c) :
              …
         else:
              …
```

Elif

- A special kind of nesting is the chain of if-else-if-else-… statements
- Can be written elegantly using if-elif-..-else

```
if cond1:
         s1
else:
    if cond2:
        s2
    else:
        if cond3:
            s3
        else:
             …
```

```
if cond1:
         s1
elif cond2:
    s2
elif cond3:
    s3
elif  …
else
     last-block-of-stmt
```

Summary of if, if-else

- if-else, nested if's, elif.
- Multiple ways to solve a problem
  - issues of readability, maintainability
  - and efficiency

Class Quiz

- What is the value of expression:

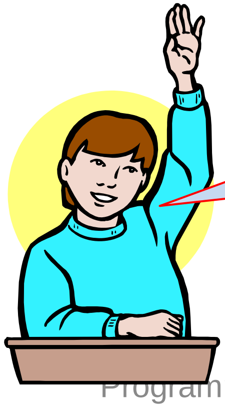
```
(5<2) and (3/0 > 1)
```

a) Run time crash/error

b) I don’t know / I don’t care

c) False

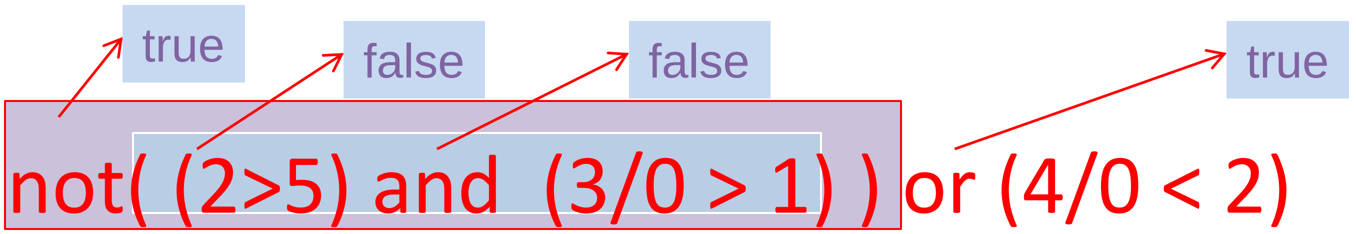
d) True



The correct answer is **False**

Short-circuit Evaluation

- Do not evaluate the second operand of binary short-circuit logical operator if the result can be deduced from the first operand
  - Also applies to nested logical operators



true

false

false

true

not((2>5) and (3/0 > 1)) or (4/0 < 2)

Evaluates to true

3 Factors for Expr Evaluation

- **Precedence**
  - Applied to two different class of operators
  - + and *, - and *, and and or, …
- **Associativity**
  - Applied to operators of same class
  - * and *, + and -, * and /, …
- **Order**
  - Precedence and associativity **identify the operands** for each operator
  - **Not which operand is evaluated first**
  - Python evaluates expressions from left to right
  - While evaluating an assignment, the right-hand side is evaluated before the left-hand side.

Class Quiz

- What is the output of the following program:

```
y = 0.1*3
if y != 0.3:
    print ('Launch a Missile')
else:
    print ("Let's have peace")
```

Launch a Missile

Caution about Using Floats

- Representation of *real numbers* in a computer can not be exact
  - Computers have limited memory to store data
  - *Between any two distinct real numbers, there are infinitely many real numbers.*
- On a typical machine running Python, there are 53 bits of precision available for a Python float

Caution about Using Floats

- The value stored internally for the decimal number 0.1 is the binary fraction

0.00011001100110011001100110011001100110011001100110011010

- Equivalent to decimal value

0.1000000000000000055511151231257827021181583404541015625

- Approximation is similar to decimal approximation 1/3 = 0.333333333...
- No matter how many digits you use, you have an approximation

Comparing Floats

- Because of the approximations, comparison of floats is not exact.
- **Solution?**
- Instead of

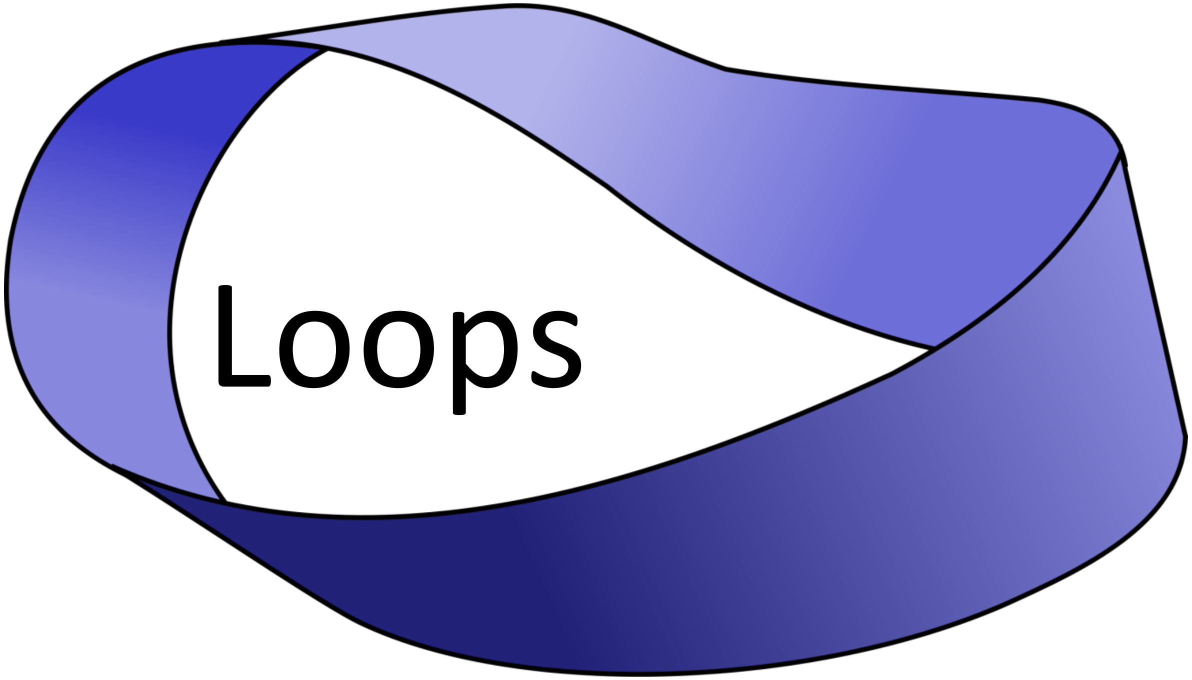
x == y

use

abs(x-y) <= epsilon

where epsilon is a suitably chosen small value

Programming using Python



Loops

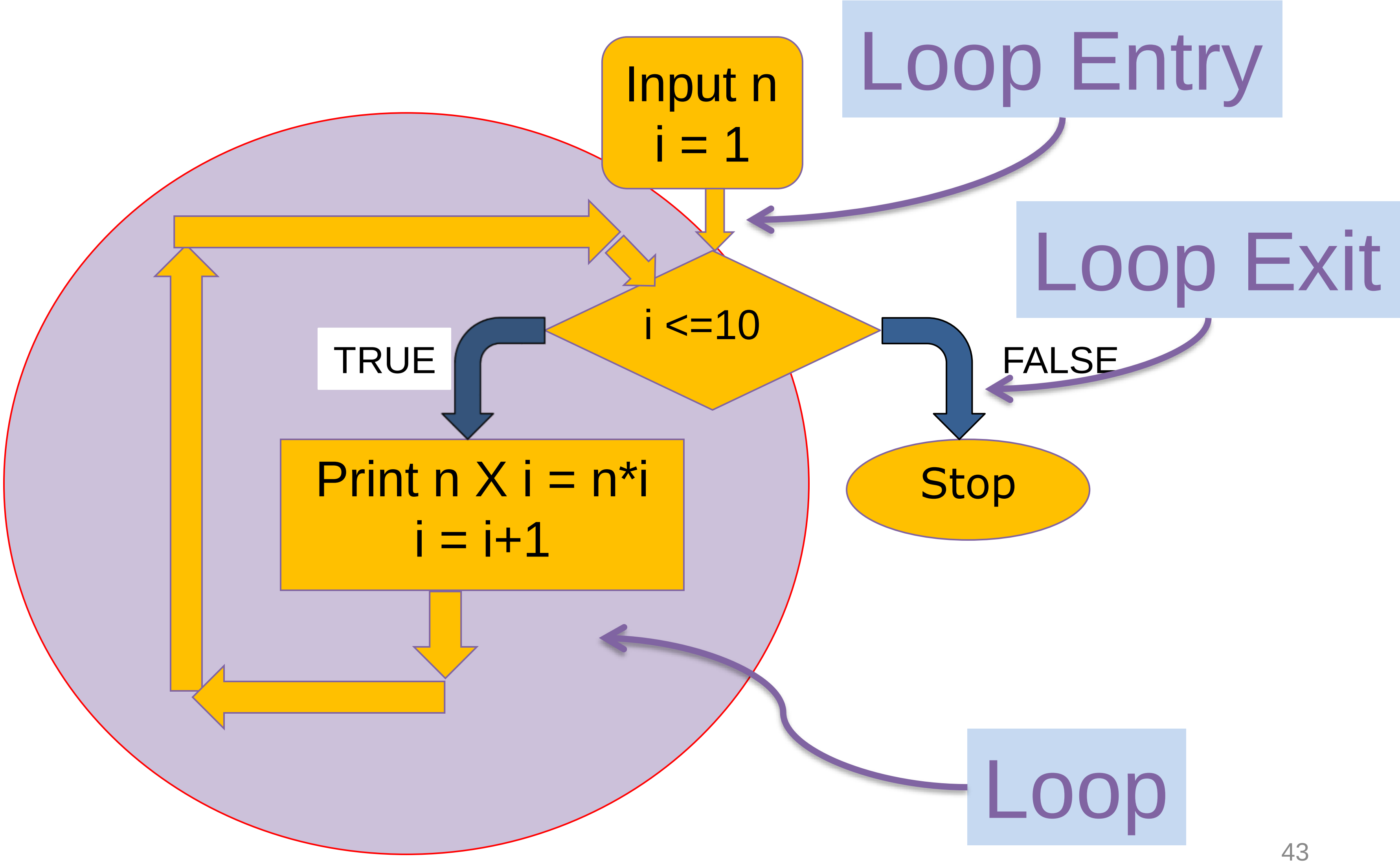
Printing Multiplication Table

5	X	1	=	5
5	X	2	=	10
5	X	3	=	15
5	X	4	=	20
5	X	5	=	25
5	X	6	=	30
5	X	7	=	35
5	X	8	=	40
5	X	9	=	45
5	X	10	=	50

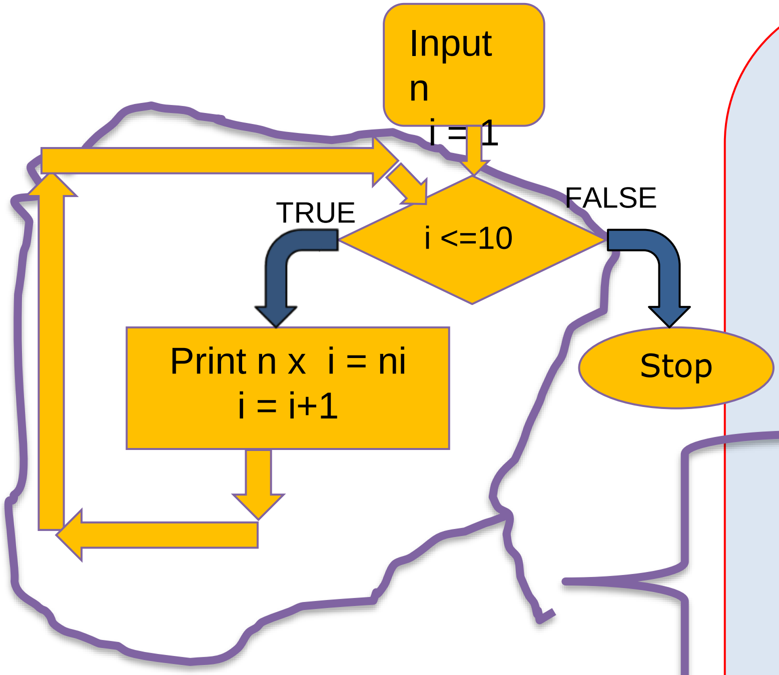
Program…

```
n = int(input('Enter a number: '))
print (n, 'X', 1, '=', n*1)
print (n, 'X', 2, '=', n*2)
print (n, 'X', 3, '=', n*3)
print (n, 'X', 4, '=', n*4)
print (n, 'X', 5, '=', n*5)
print (n, 'X', 6, '=', n*6)

….
```

Printing Multiplication Table



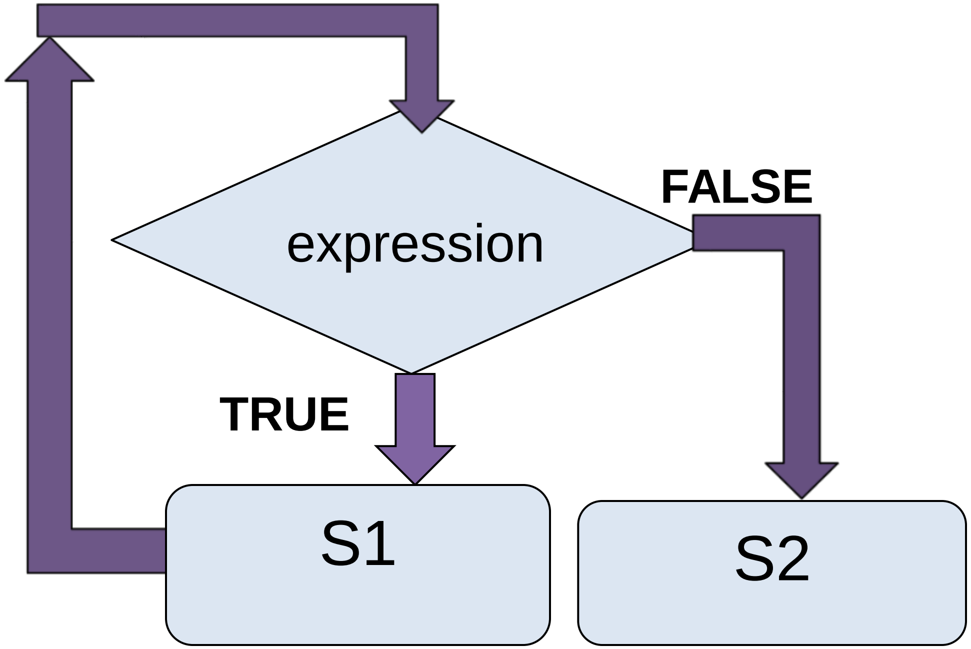
Printing Multiplication Table



```
n = int(input('n=? '))
i = 1

while (i <= 10):
    print (n ,'X', i, '=', n*i)
    i = i + 1
print ('done‘)
```

While Statement

```
while (expression):
        S1
S2
```



1. Evaluate expression
2. If TRUE then
    a) execute statement1
    b) goto step 1.
3. If FALSE then execute statement2.

For Loop

- Print the sum of the reciprocals of the first 100 natural numbers.

```
rsum=0.0# the reciprocal sum

# the for loop
for i in range(1,101):
    rsum = rsum + 1.0/i
print ('sum is', rsum)
```

For loop in Python

- General form

```
for variable in sequence:
    stmt
```

range

- range(s, e, d)
  - generates the list:

    [s, s+d, s+2*d, …, s+k*d]

    where s+k*d < e <= s+(k+1)*d
- range(s, e) is equivalent to range(s, e, 1)
- range(e) is equivalent to range(0, e)

Exercise: What if d is negative? Use python interpreter to find out.

Quiz

- What will be the output of the following program

```
# print all odd numbers < 10
i = 1
while i <= 10:
    if i%2==0: # even
        continue
    print (i, end=' ')
    i = i+1
```

Continue and Update Expr

- Make sure continue does not bypass update-expression for while loops

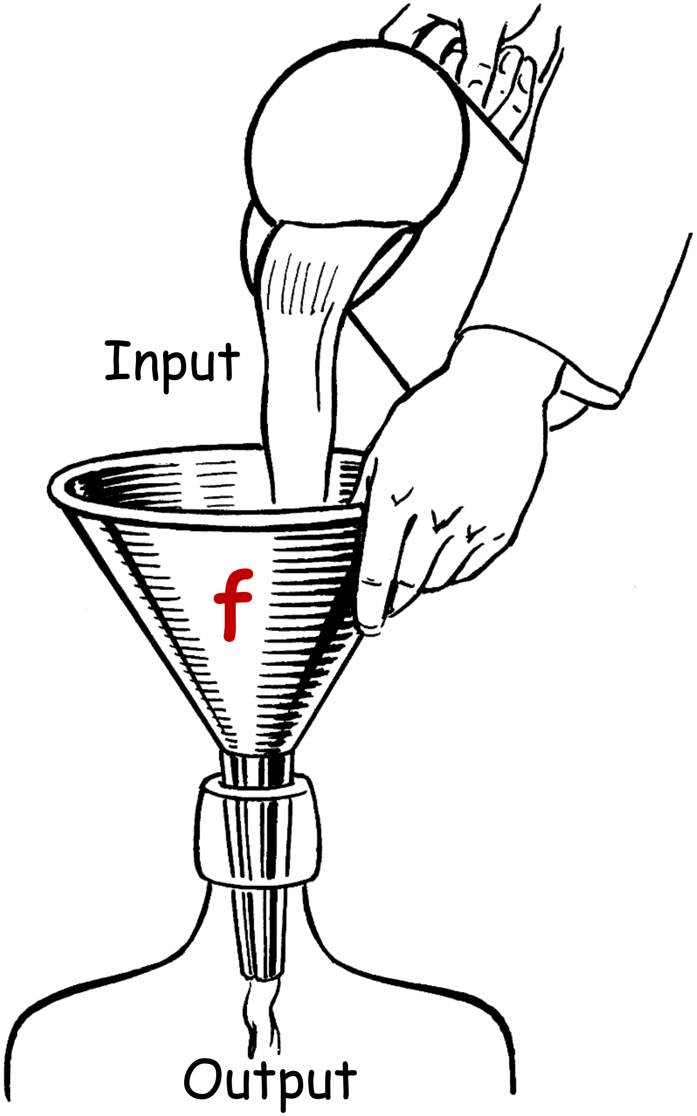
```
# print all odd numbers < 10
i = 1
while i <= 10:
   if i%2==0: # even
      continue
   print (i, end=' ')
   i = i+1
```

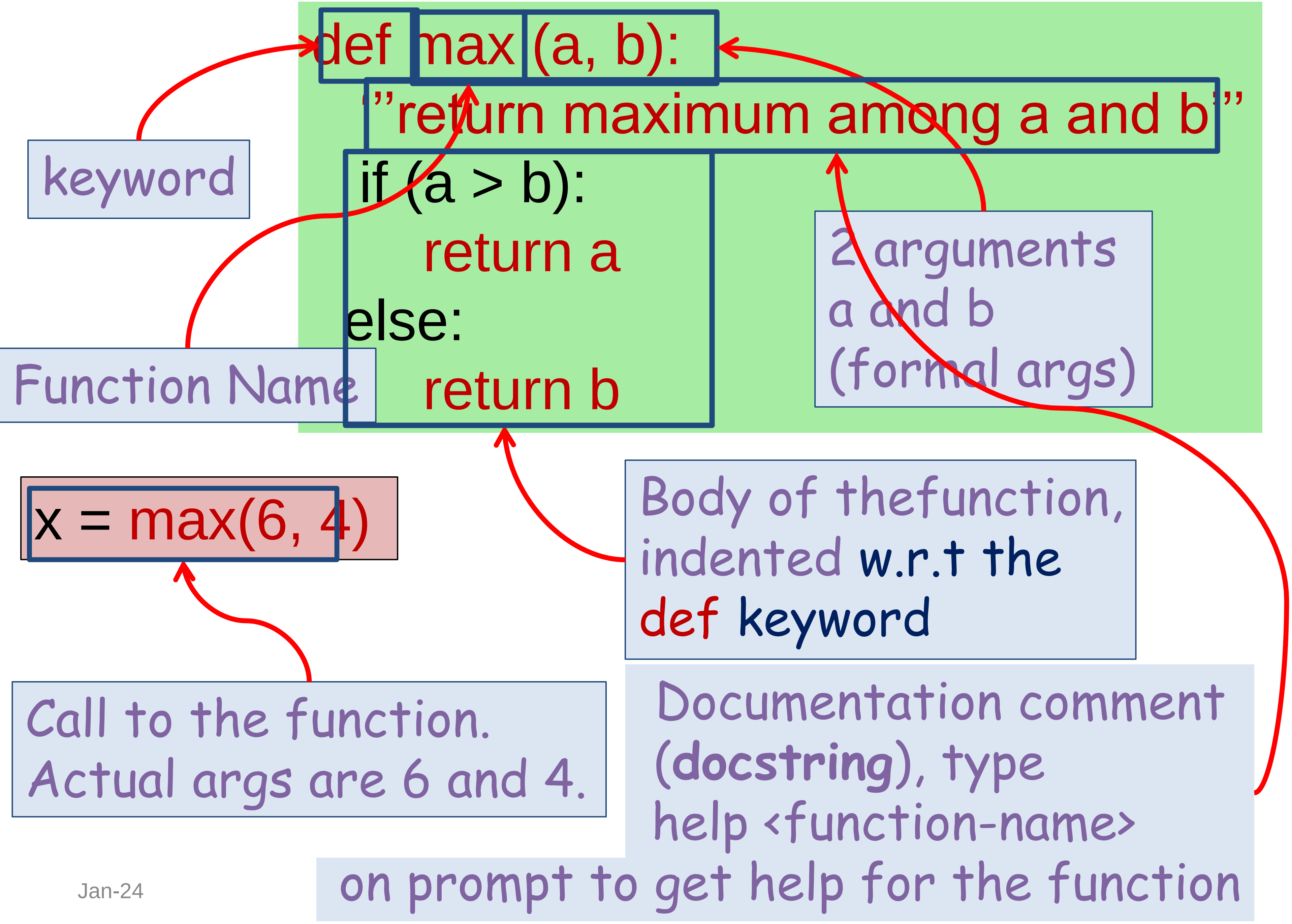
i is not incremented when even number encountered. Infinite loop!!

Programming using Python

f(unctions)

Parts of a function





```
def max (a, b):
   ‘”return maximum among a and b’”
   if (a > b):
       return a
   else:
       return b
```

keyword

Function Name

2 arguments a and b (formal args)

Body of thefunction, indented w.r.t the def keyword

```
x = max(6, 4)
```

Call to the function. Actual args are 6 and 4.

Documentation comment (**docstring**), type help <function-name> on prompt to get help for the function

```
def max (a, b):
    “‘return maximum among a and b’”
    if (a > b):
        return a
   else:
        return b
```

```
In[3] : help(max)
Help on function max in module __main__:

max(a, b)
    return maximum among a and b
```

Keyword Arguments

```
def printName(first, last, initials) :
    if initials:
        print (first[0] + '. ' + last[0] + '.')
    else:
        print (first, last)
```

Note use of [0] to get the first character of a string. More on this later.

Call	Output
printName('Acads', 'Institute', False)	Acads Institute
printName('Acads', 'Institute', True)	A. I.
printName(last='Institute', initials=False, first='Acads')	Acads Institute
printName('Acads', initials=True, last='Institute')	A. I.

Keyword Arguments

- Parameter passing where formal is bound to actual using formal's name
- Can mix keyword and non-keyword arguments
  - All non-keyword arguments precede keyword arguments in the call
  - Non-keyword arguments are matched by position (order is important)
  - Order of keyword arguments is not important

Default Values

```
def printName(first, last, initials=False) :
   if initials:
      print (first[0] + '. ' + last[0] + '.')
   else:
      print (first, last)
```

Note the use of “default” value

Call	Output
printName('Acads', 'Institute')	Acads Institute
printName(first='Acads', last='Institute', initials=True)	A. I.
printName(last='Institute', first='Acads')	Acads Institute
printName('Acads', last='Institute')	Acads Institute

Default Values

- Allows user to call a function with fewer arguments
- Useful when some argument has a fixed value for most of the calls
- All arguments with default values must be at the end of argument list
  - non-default argument can not follow default argument

Globals

- Globals allow functions to communicate with each other indirectly
  - Without parameter passing/return value
- Convenient when two seemingly “far-apart” functions want to share data
  - No *direct* caller/callee relation
- If a function has to update a global, it must re-declare the global variable with global keyword.

Globals

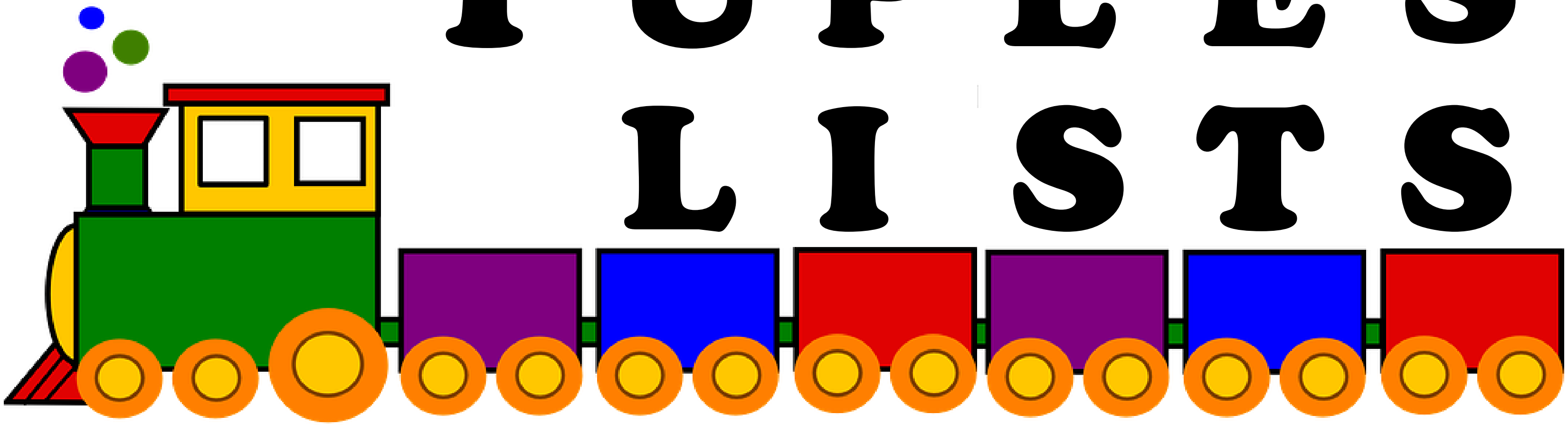
```
PI = 3.14
def perimeter(r):
   return 2 * PI * r
def area(r):
   return PI * r * r
def update_pi():
   global PI
   PI = 3.14159
```

```
>>> print(area (100))
31400.0
>>> print(perimeter(10))
62.800000000000004
>>> update_pi()
>>> print(area(100))
31415.999999999996
>>> print(perimeter(10))
62.832
```

defines PI to be of float type with value 3.14. PI can be used across functions. Any change to PI in update_pi will be visible to all due to the use of global.

Programming with Python

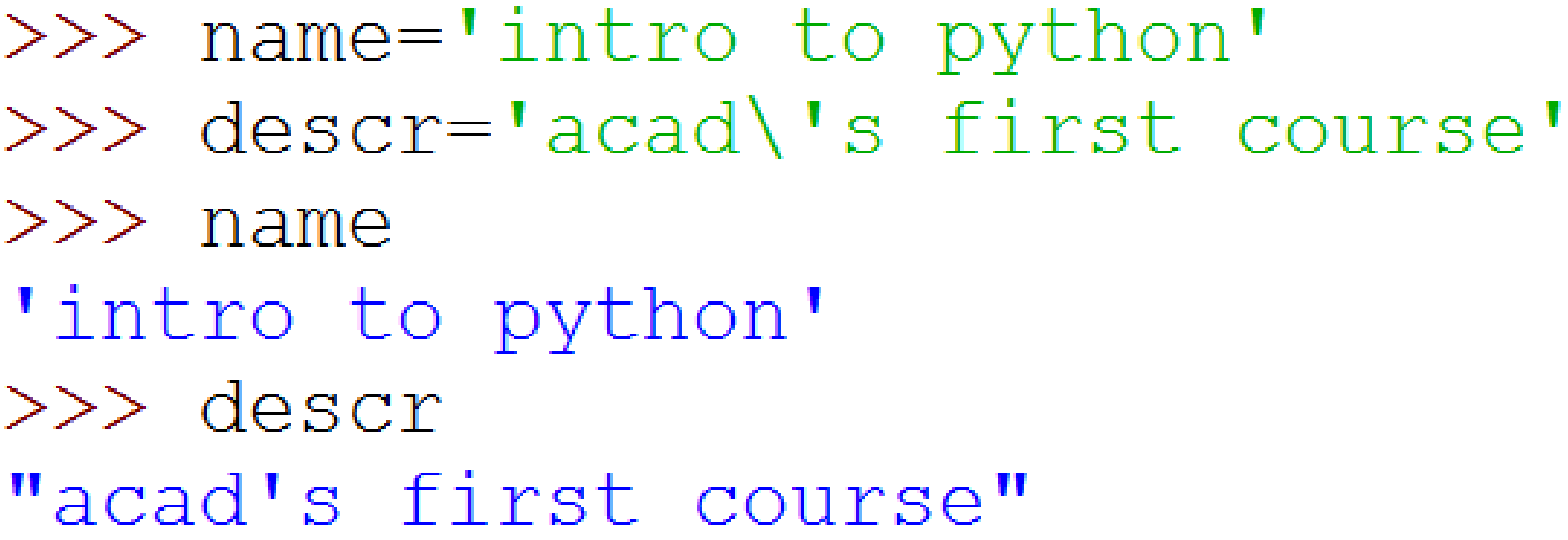
STRINGS
TUPLES
LISTS



Strings

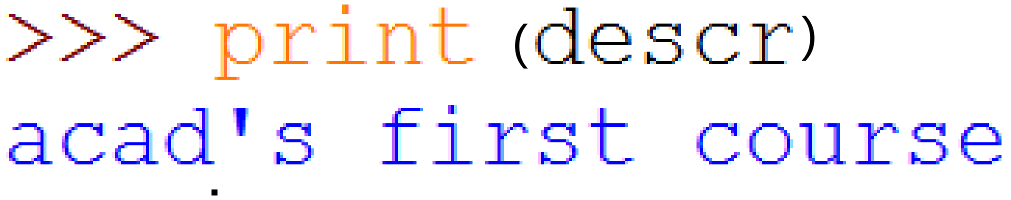
- Strings in Python have type str
- They represent sequence of characters
  - Python does not have a type corresponding to character.
- Strings are enclosed in single quotes(') or double quotes(“)
  - Both are equivalent
- Backslash (\) is used to escape quotes and special characters

Strings



```
>>> name='intro to python'
>>> descr='acad\'s first course'
>>> name
'intro to python'
>>> descr
"acad's first course"
```

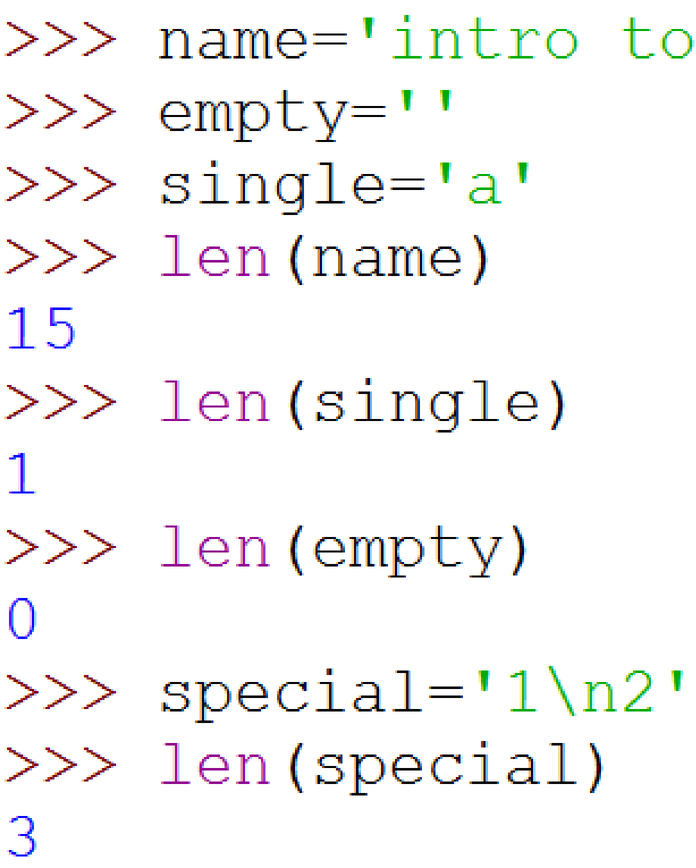
- More readable when print is used



```
>>> print (descr)
acad's first course
```

Length of a String

- len function gives the length of a string



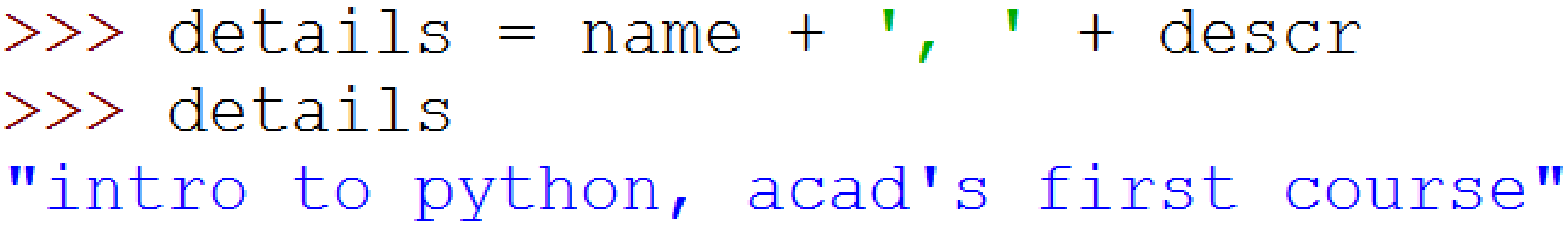
```
>>> name='intro to python'
>>> empty=''
>>> single='a'
>>> len(name)
15
>>> len(single)
1
>>> len(empty)
0
>>> special='1\n2'
>>> len(special)
3
```

\n is a **single** character: the special character representing newline

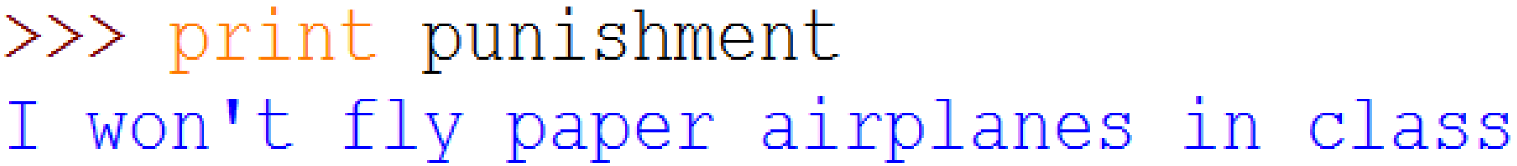
Concatenate and Repeat

- In Python, + and * operations have special meaning when operating on strings
  - + is used for concatenation of (two) strings
  - * is used to repeat a string, an int number of time
  - Function/Operator Overloading

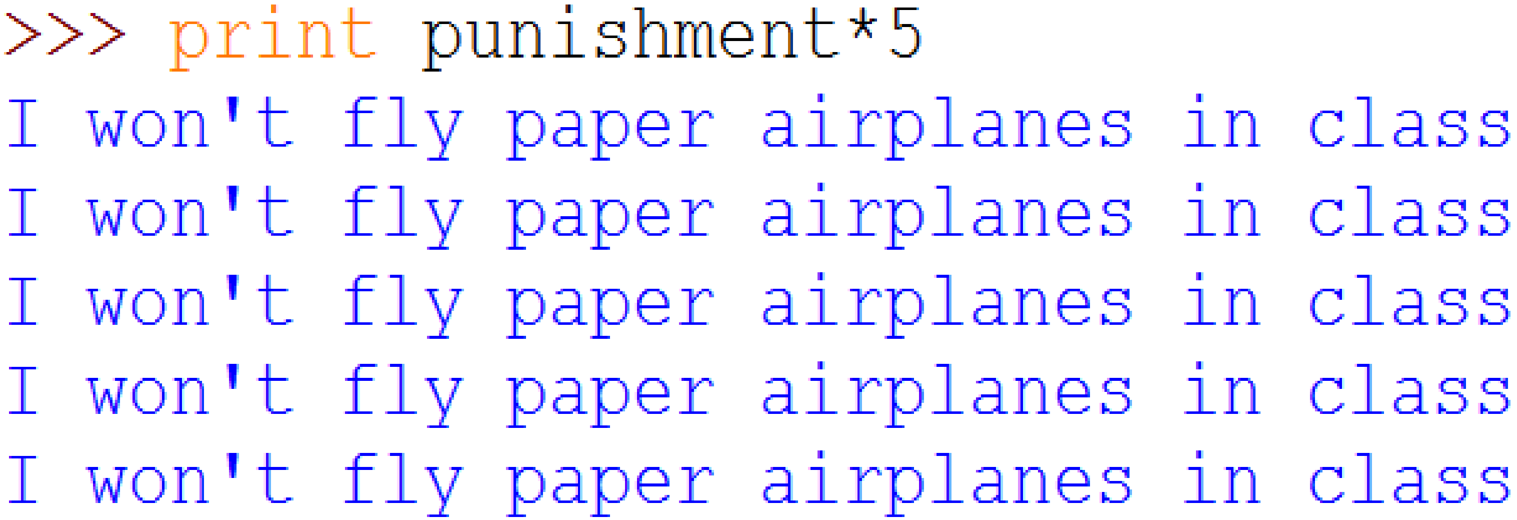
Concatenate and Repeat



```
>>> details = name + ', ' + descr
>>> details
"intro to python, acad's first course"
```



```
>>> print punishment
I won't fly paper airplanes in class
```

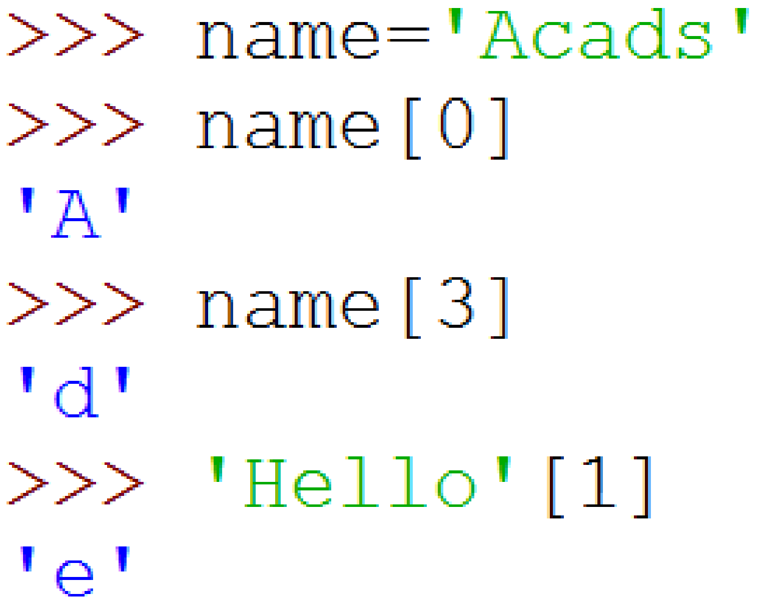


```
>>> print punishment*5
I won't fly paper airplanes in class
I won't fly paper airplanes in class
I won't fly paper airplanes in class
I won't fly paper airplanes in class
I won't fly paper airplanes in class
```

Note: Put round brackets after print

Indexing

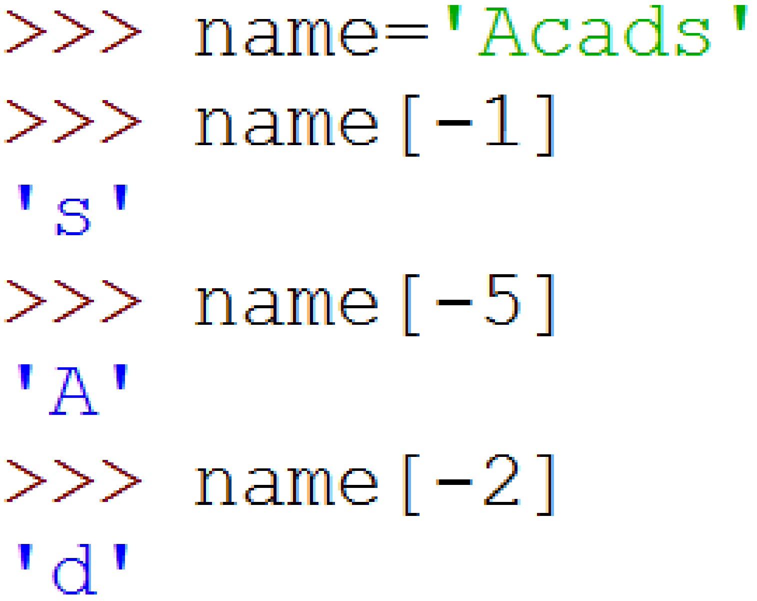
- Strings can be indexed
- First character has index 0



```
>>> name='Acads'
>>> name[0]
'A'
>>> name[3]
'd'
>>> 'Hello'[1]
'e'
```

Indexing

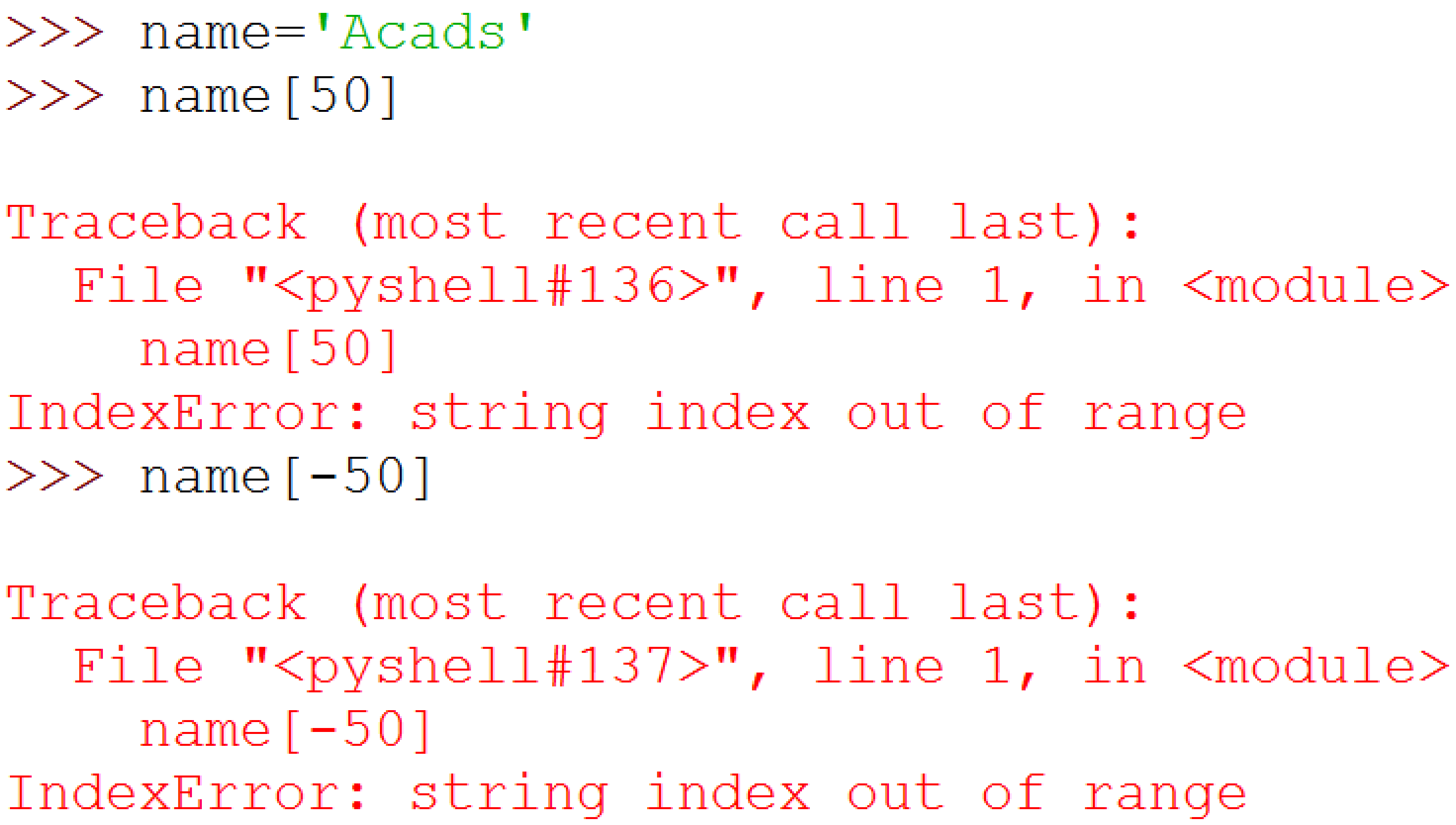
- Negative indices start counting from the right
- Negatives indices start from -1
- -1 means last, -2 second last, ...



```
>>> name='Acads'
>>> name[-1]
's'
>>> name[-5]
'A'
>>> name[-2]
'd'
```

Indexing

- Using an index that is too large or too small results in “index out of range” error

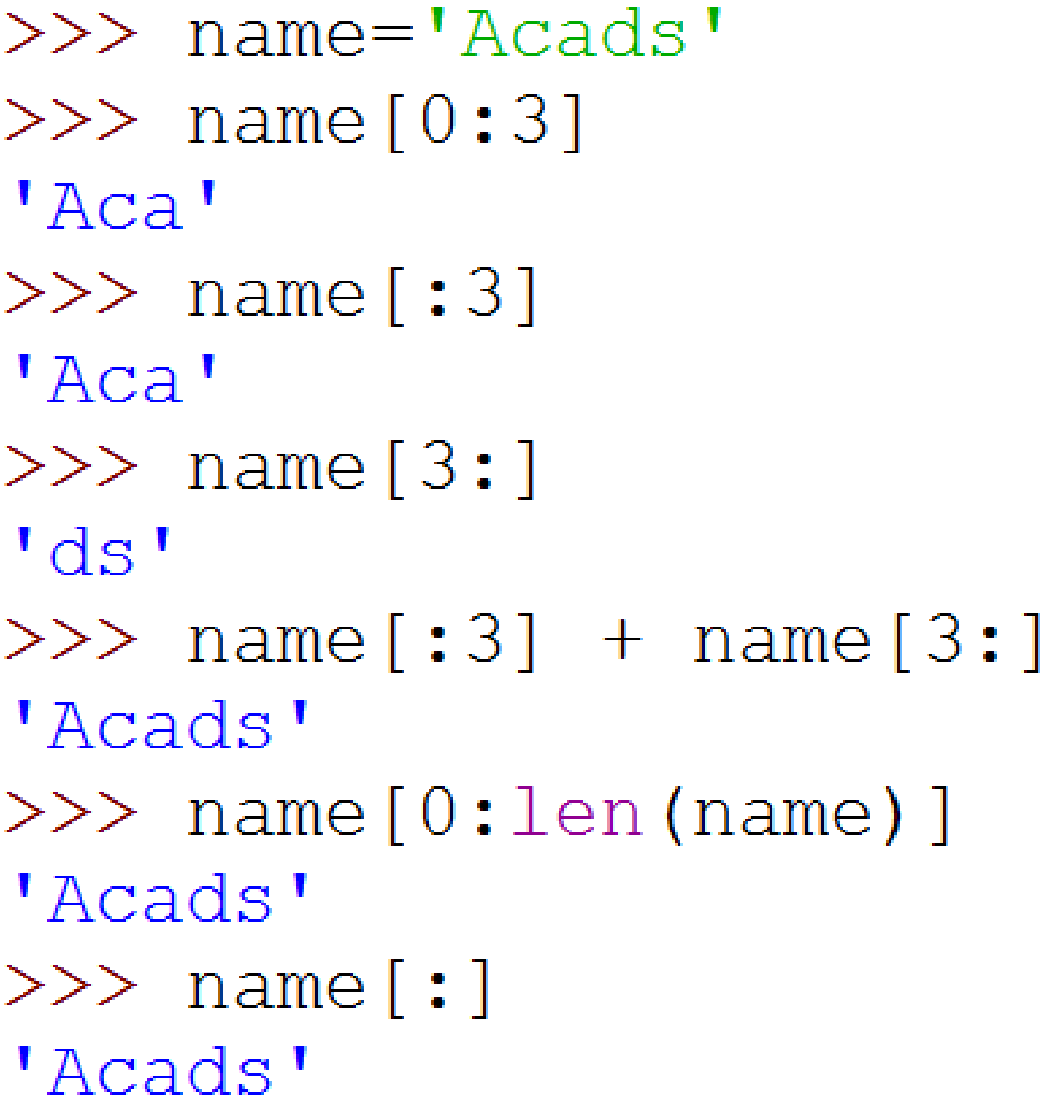


```
>>> name='Acads'
>>> name[50]

Traceback (most recent call last):
  File "<pyshell#136>", line 1, in <module>
    name[50]
IndexError: string index out of range
>>> name[-50]

Traceback (most recent call last):
  File "<pyshell#137>", line 1, in <module>
    name[-50]
IndexError: string index out of range
```

Slicing

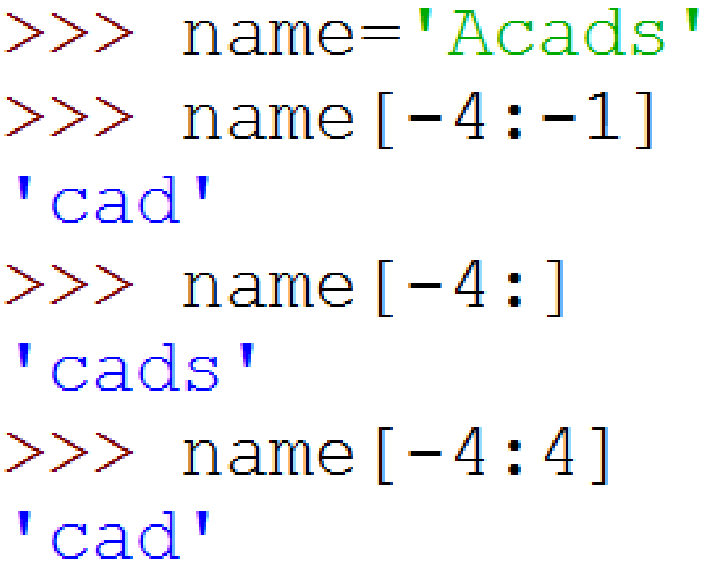
- To obtain a substring
- s[start:end] means substring of s starting at index start and ending at index end-1
- s[0:len(s)] is same as s
- Both start and end are optional
  - If start is omitted, it defaults to 0
  - If end is omitted, it defaults to the length of string
- s[:] is same as s[0:len(s)], that is same as s

Slicing



```
>>> name='Acads'
>>> name[0:3]
'Aca'
>>> name[:3]
'Aca'
>>> name[3:]
'ds'
>>> name[:3] + name[3:]
'Acads'
>>> name[0:len(name)]
'Acads'
>>> name[:]
'Acads'
```

More Slicing



```
>>> name='Acads'
>>> name[-4:-1]
'cad'
>>> name[-4:]
'cads'
>>> name[-4:4]
'cad'
```

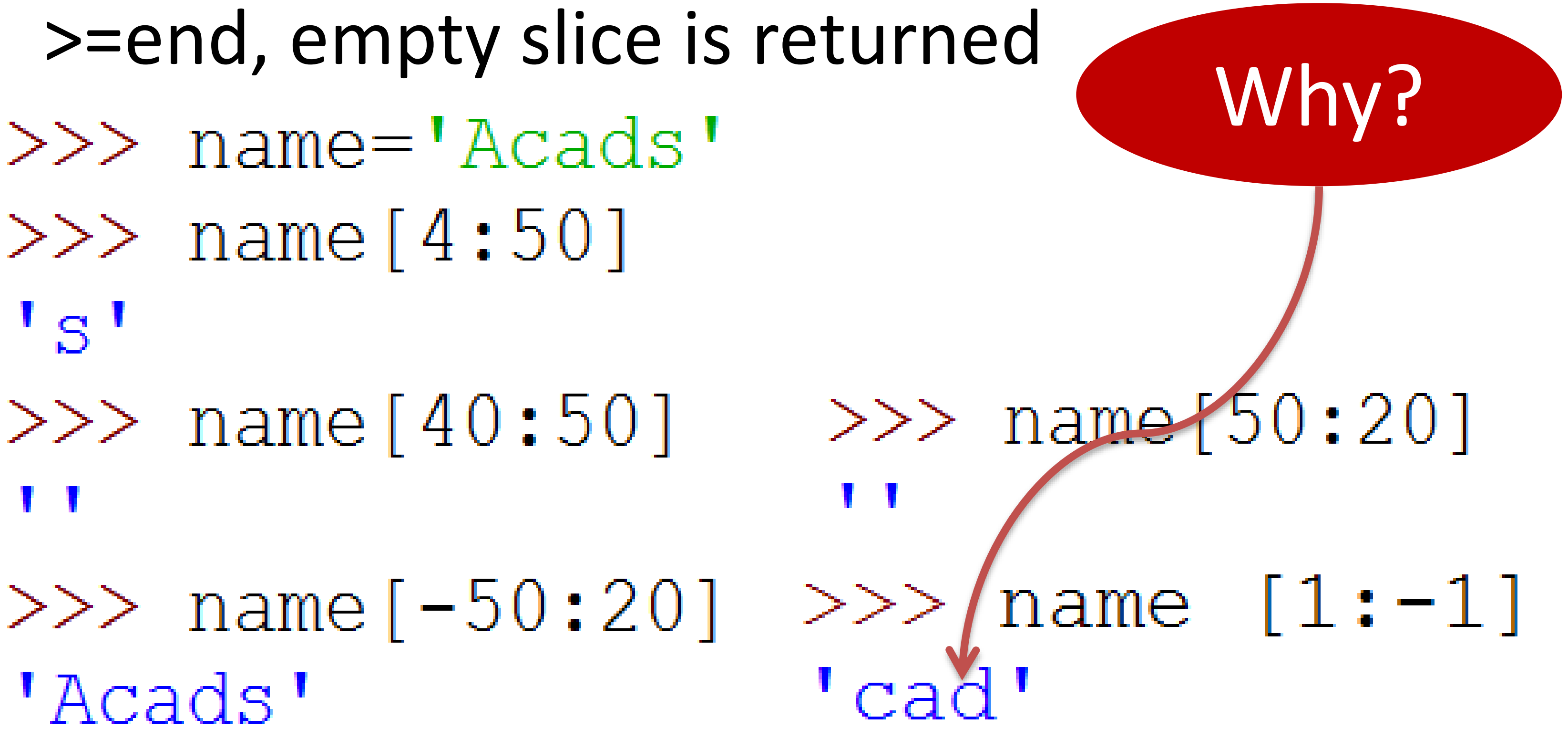
Understanding Indices for slicing

A	c	a	d	s	
0	1	2	3	4	5
-5	-4	-3	-2	-1	

Out of Range Slicing

A	c	a	d	s
0	1	2	3	4
-5	-4	-3	-2	-1

- Out of range indices are ignored for slicing
- when start and end have the same sign, if start >=end, empty slice is returned



```
>>> name='Acads'
>>> name[4:50]
's'
>>> name[40:50]
''
>>> name[-50:20]
'Acads'
```

```
>>> name[50:20]
''
>>> name [1:-1]
'cad'
```

Why?