Iteration & Strings

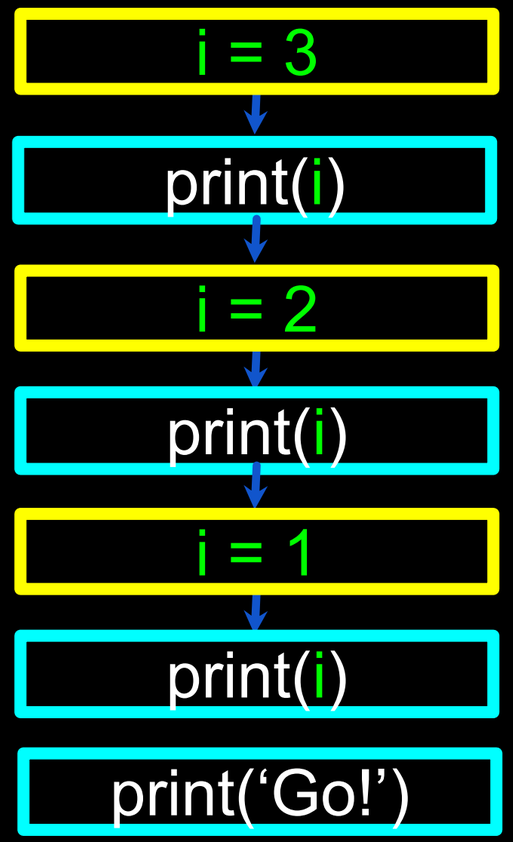
Lecture 4

A definite Loop

a sequence of items

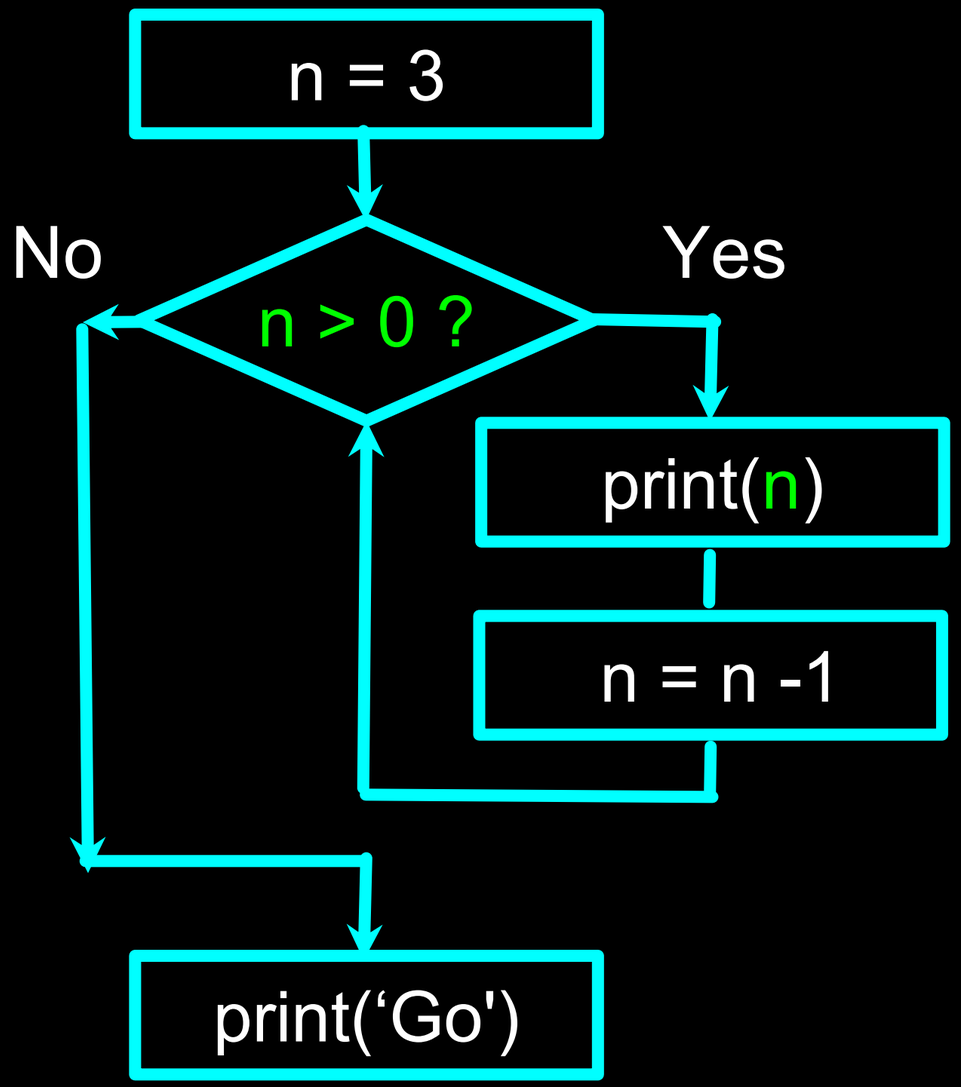
```
for i in [3, 2, 1] :
    print(i)
print('Go!')
```

```
3
2
1
Go!
```



Indefinite Loop with `while`

while loop

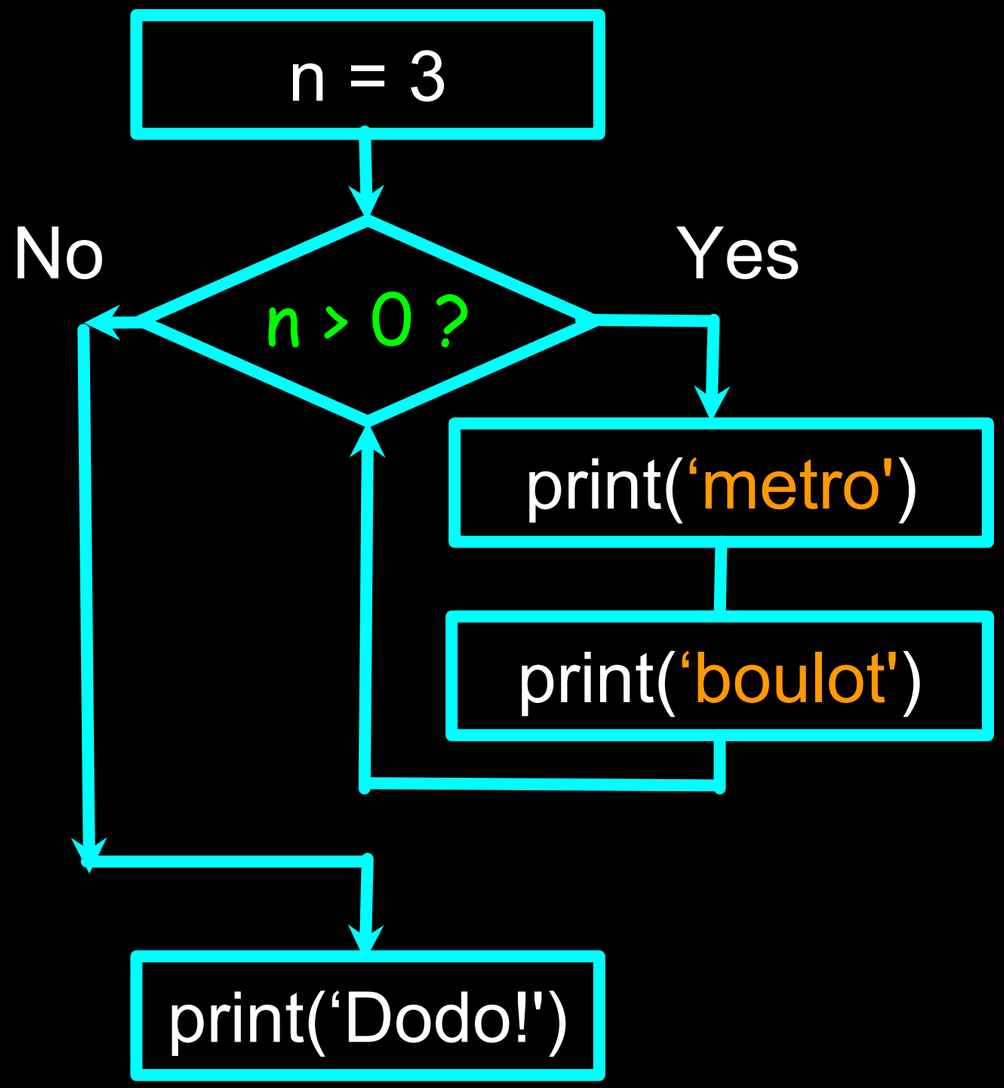


Program:

```
n = 3
while n > 0 :
    print(n)
    n = n – 1
print(‘Go!')
print(n)
```

Output:

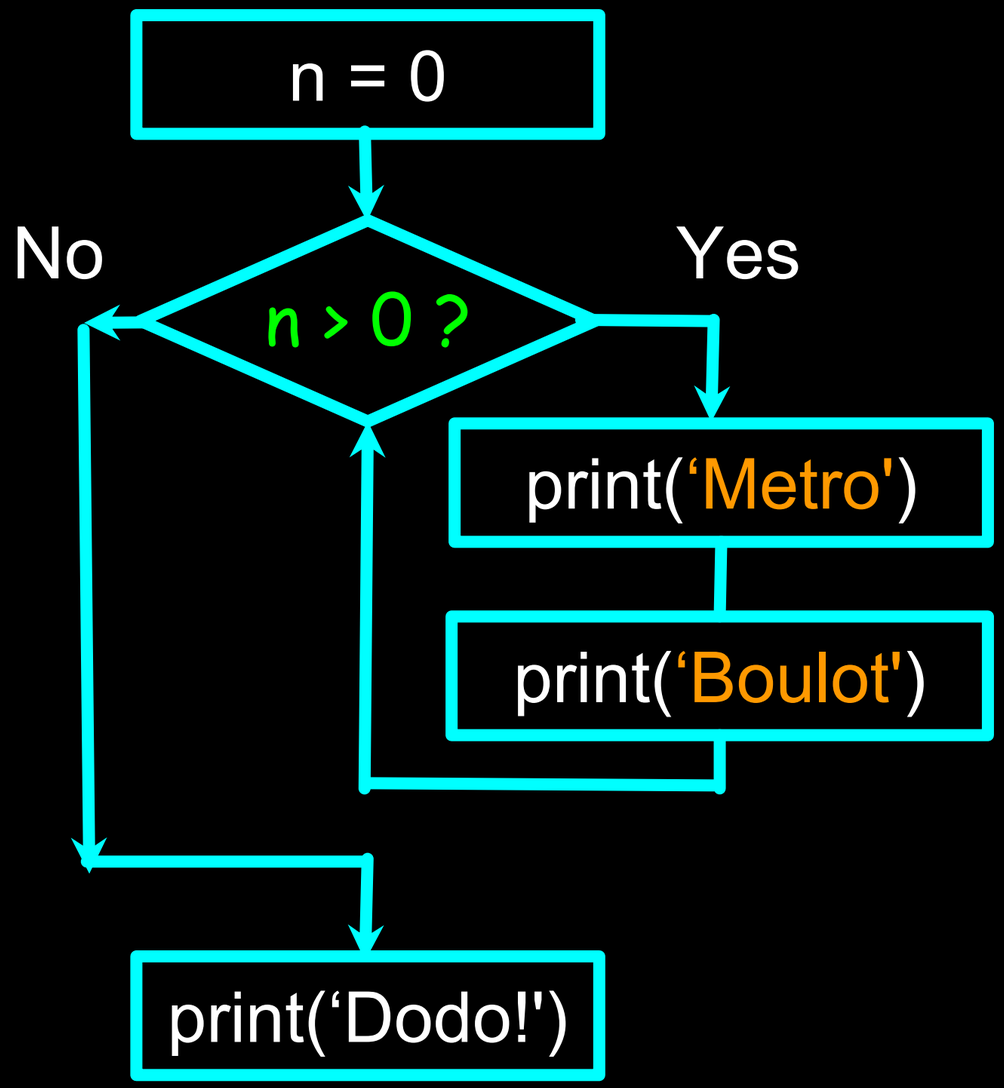
3
2
1
Go!
0



An Infinite Loop

```
n = 3
while n > 0 :
    print(‘metro')
    print(‘boulot')
print(‘Dodo!’)
```

What is wrong with this loop?



Another Loop

```
n = 0
while n > 0 :
    print(‘Metro')
    print(‘Boulot')
print(‘Dodo!')
```

What is this loop doing?

Checking vowel letters

```
word = “alphebet”
vowels = “aeiou”
n = 0
while n < len(word) :
     vowels = word[n]
     if letter in vowels :
        print(letter)
     n = n + 1
print(‘Done!')
```

```
n += 1
```

? Output:

a

e

e

Done!

break & continue

Breaking Out of a Loop

- break statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

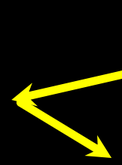
```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

Breaking Out of a Loop

- break : ends the current loop

```
name = "Marie"
for letter in name :
    if letter == ‘i' :
        break
    print(letter)
print('Done!')
```



Output:
M
a
r
Done!

Finishing an Iteration

continue: ends the current iteration

```
name = "Marie"
for letter in name :
    if letter == ‘i' :
        continue
    print(letter)
print('Done!')
```

Output:

```
M
a
r
e
Done!
```

Working with `string`

String Data Type

- A string is a sequence of characters
- A string uses quotes
  'Hello' or "Hello"
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using int()

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

Special characters: `\s` `\n` `\t`

Non-printable Characters

Represented with backslash notation

- \s space
- \t tab
- \n new line
- still one character - not two

```
>>> stuff = 'Hello\sWorld!'
>>> print(stuff)
Hello World!
>>> stuff = 'Hello\tWorld!'
>>> print(stuff)
Hello    World!
>>> stuff = ‘X\nY\nZ'
>>> print(stuff)
X
Y
Z
>>> len(stuff)
5
```

Looking Inside Strings

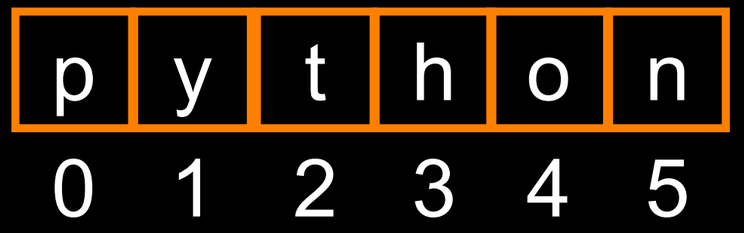
Get a single character : `string[id]`

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

Get a sequence of elements

```
>>> course = ‘python'
>>> course[1:3]
‘yt’
>>> course[:4]
‘pyth’
>>> course[3:]
‘hon’
>>> t[:]
‘python’
```




```
>>> print(course[6])
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
IndexError: string index out
of range
>>>
```

Looping Through Strings

- while statement
- iteration variable
- len function

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

```
0 b
1 a
2 n
3 a
4 n
5 a
```

Looping Through Strings

- for loop: much more elegant

```
fruit = 'banana'
for letter in fruit:
    print(letter)
```

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

```
b
a
n
a
n
a
```

Looping and Counting

?

```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
```

More String Operations

Operators: `in` & `not in`

`Lists:` list element?

```
>>> some = [1, 9, 21, 10]
>>> 9 in some
True
>>> 20 not in some
True
```

string: string element?

```
>>> ss = ‘pencil’
>>> “s” in ss
False
>>> ‘e’ not in ss
False
>>> ‘pen' in ss
True
```

String methods

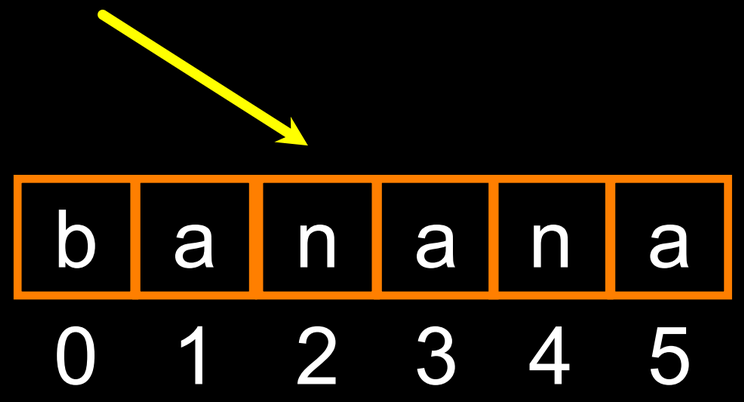
- string.method()
  appending the method to the string variable
- method do not modify the original string, instead they return a new string

```
>>> greet = ‘HELLO BOB'
>>> zap = greet.lower()
>>> print(zap)
hello bob
>>> print(greet)
HELLO BOB
>>> print(zap.upper())
HELLO BOB
```

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', '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



- find(): search for a substring within another string
- find() finds the first occurrence
- If not found, find() returns -1

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

Parsing and Extracting

21 31

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```

Search and Replace

- replace(): replaces all occurrences of the search string with the replacement string

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

Stripping Whitespace

- lstrip() and rstrip() remove whitespace at the left or right
- strip() removes both beginning and ending whitespace

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

split() Pattern

- Split strings according to delimiters (space if not provided)
- Returns a list of substrings

```
>>> line = “Good morning, Bob!”
>>> print(line.split())
>>> print(line.split(‘,’))
>>> print(line.split(‘!’)))
```

```
['Good', 'morning,', 'Bob!']
['Good morning', ' Bob!']
['Good morning, Bob', '']
```

`split()` Pattern

- Split a line according to delimiters (space if not provided)
- Returns a list of substrings

```
From alice.dupont@etu.u-paris.fr Sat Jan  5 09:14:16 2021
```

```
words = line.split()
email = words[1]
pieces = email.split('@')
print(pieces[1])
```

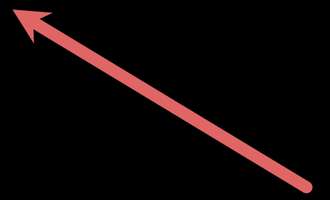
```
alice.dupont@etu.u-paris.fr
['alice.dupont', 'etu.u-paris.fr']
'etu.u-paris.fr'
```

Prefixes

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

```
x=input()
if type(int(x)) == int:
    print(x, “is integer”)
elif type(float(x)) == float:
    print(x, “is float”)
elif type(x) == str:
    print(x, “is sring)
else:
    print(“***”)
```

Traceback (most recent call last):
line 2, in <module> istr = int(astr)
ValueError: invalid literal for int() with base 10: 'Hello Bob'



All Done

The program stops here

```
x=input()
if type(int(x)) == int:
```

Traceback (most recent call last):
line 2, in <module> istr = int(astr)
ValueError: invalid literal for int() with base 10: 'Hello Bob'

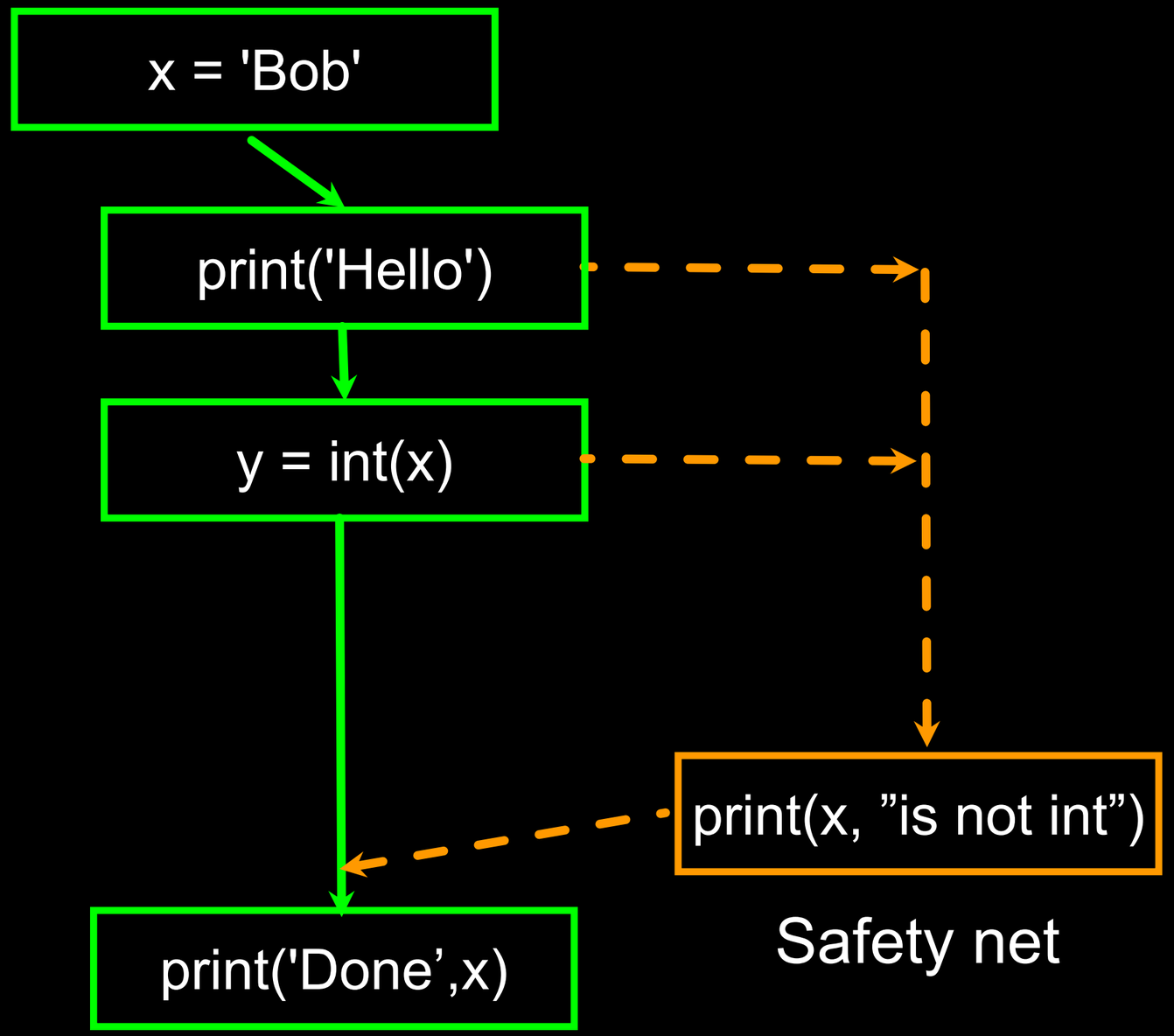
All Done

try / except

```
x = 'Bob'
try:
    print('Hello')
    y = int(x)
except:
    print(x,“is not int”)

print('Done’, x)
```

```
Output:
Bob is not int.
Done Bob
```

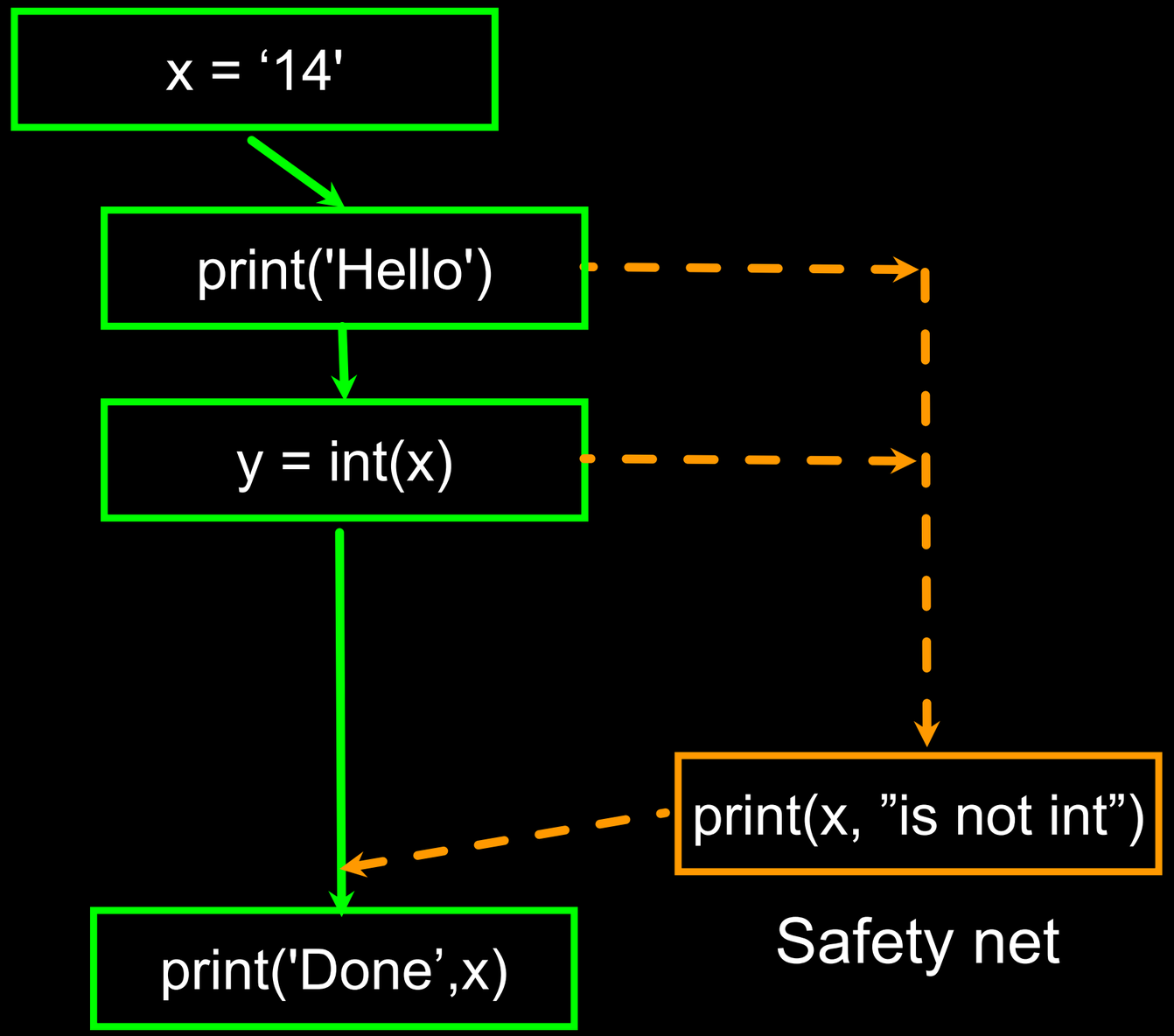


try / except

```
x = ‘14'
try:
    print('Hello')
    y = int(x)
except:
    print(x,“is not int”)

print('Done’, x)
```

```
Output:
Hello
Done Bob
```



Visualize Blocks

```
x = 4

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

print('All done')
```