Imperative Programming

- Objects and Classes
- Python Standard Libraries
- Python Programs
- Interactive Input/Output
- One-Way and Two-Way `if` Statements
- `for` Loops

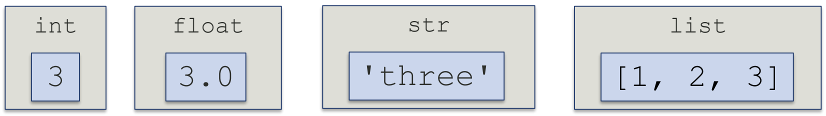
Objects and classes

In Python, every value, whether a simple integer value like `3` or a more complex value, such as the list `['hello', 4, 5]` is stored in memory as an object.

Every object has a value and a type;

It is the object that has a type, not the variable!

```
>>> a = 3
>>> b = 3.0
>>> c = 'three'
>>> d = [1, 2, 3]
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> type(d)
<class 'list'>
>>> a = []
>>> type(a)
<class 'list'>
```



An object's type determines what values it can have and how it can be manipulated

Terminology: object X is of type `int` = object X belongs to class `int`

Values of number types

An object's type determines what values it can have and how it can be manipulated

An object of type `int` can have, essentially, any integer number value

The value of an object of type `float` is represented in memory using 64 bits
- i.e., 64 zeros and ones

This means that only 2^{64} real number values can be represented with a `float` object; all other real number values are just approximated

```
>>> 0
0
>>> 2**1024
1797693134862315907729305
1907890247336179769789423
0657273430081157732675805
5009631327084773224075360
2112011387987139335765878
9768814416622492847430639
4741243777678934248654852
7630221960124609411945308
2952085005768838150682342
4628814739131105408272371
6335051068458629823994724
5938479716304835356329624
224137216
>>> 0.0
0.0
>>> 2.0**1024
Traceback (most recent
call last):
  File "<pyshell#38>",
line 1, in <module>
    2.0**1024
OverflowError: (34,
'Result too large')
>>> 2.0**(-1075)
0.0
```

Operators for number types

An object's type determines what values it can have and how it can be manipulated

We already saw the operators that are used to manipulate number types
- algebraic operators +, -, *, /, //, %, **, abs()
- comparison operators >, <, ==, !=, <=, >=, …

Parentheses and precedence rules determine the order in which operators are evaluated in an expression

higher precedence

Operator
`[…]`
`x[]`
`**`
`+x, -x`
`*, /, //, %`
`+, -`
`in, not in`
`<,>,<=,>=,==,!=`
`not x`
`and`
`or`

lower precedence

lower precedence

Object constructors

```
>>> x = 3
>>> x
3
>>> x = int(3)
>>> x
3
>>> x = int()
>>> x
0
>>> y = float()
>>> y
0.0
>>> s = str()
>>> s
''
>>> lst = list()
>>> lst
[]
>>>
```

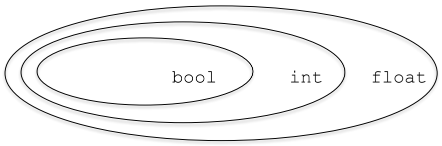
An assignment statement can be used to create an integer object with value 3

- The type of the object is implicitly defined

The object can also be created by explicitly specifying the object type using a constructor function

- `int()`: integer constructor (default value: `0`)
- `float()`: Float constructor (default value: `0.0`)
- `str()`: string constructor (default value: empty string `''`)
- `list()`: list constructor (default value: empty list `[]`)

Type conversion



Implicit type conversion

- When evaluating an expression that contains operands of different type, operands must first be converted to the same type
- Operands are converted to the type that "contains the others"

Explicit type conversion

- Constructors can be used to explicitly convert types

```
>>> str(345)
'345'
>>> str(34.5)
'34.5'
>>>
```

`int()` creates an `int` object

- from a `float` object, by removing decimal part
- from a `str` object, if it represents an integer

`float()` creates a `float` object

- from an `int` object, if it is not too big
- from a `string`, if it represents a number

`str()` creates a `str` object

- the string representation of the object value

Class and class methods

Once again: In Python, every value is stored in memory as an object, every object belongs to a class (i.e., has a type), and the object's class determines what operations can be performed on it

We saw the operations that can be performed on classes `int` and `float`

The `list` class supports:

- operators such as `+`, `*`, `in`, `[]`, etc.
- methods such as `append()`, `count()`, `remove()`, `reverse()`, etc.

```
>>> fish = ['goldfish']
>>> myPets = ['cat', 'dog']
>>> fish * 3
['goldfish', 'goldfish', 'goldfish']
>>> pets = fish + myPets
>>> pets
['goldfish', 'cat', 'dog']
>>> 'frog' in pets
False
>>> pets[-1]
'dog'
>>>
```

Python Standard Library

The core Python programming language comes with functions such as `max()` and `sum()` and classes such as `int`, `str`, and `list`.

Many more functions and classes are defined in the Python Standard Library to support

- Network programming
- Web application programming
- Graphical user interface (GUI) development
- Database programming
- Mathematical functions
- Pseudorandom number generators
- Media processing, etc.

The Python Standard Library functions and classes are organized into components called modules.

Standard Library module `math`

The core Python language does not have a square root function

The square root function `sqrt()` is defined in the Standard Library module `math`

A module must be explicitly imported into the execution environment:

```
import <module>
```

The prefix math. must be present when using function sqrt()

The `math` module is a library of mathematical functions and constants

```
>>> import math
>>> math.sqrt(4)
2.0
>>> sqrt(4)
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in
<module>
    sqrt(4)
NameError: name 'sqrt' is not defined
>>> help(math)
Help on module math:
…
>>> math.cos(0)
1.0
>>> math.log(8)
2.0794415416798357
>>> math.log(8, 2)
3.0
>>> math.pi
3.141592653589793
```

Exercise

Write a Python expression that assigns to variable c

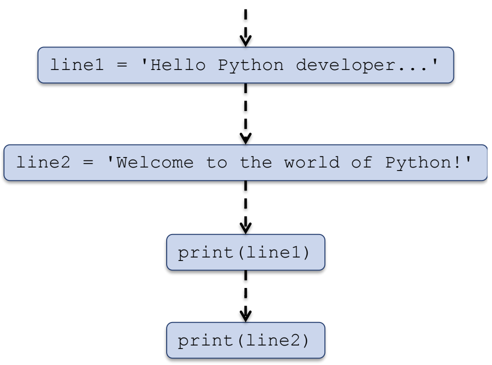
a) The length of the hypotenuse in a right triangle whose other two sides have lengths 3 and 4
b) The value of the Boolean expression that evaluates whether the length of the above hypotenuse is 5
c) The area of a disk of radius 10
d) The value of the Boolean expression that checks whether a point with coordinates (5, 5) is inside a circle with center (0,0) and radius 7.

```
>>> c = math.sqrt(3**2+4**2)
>>> c
5.0
>>> c = (math.sqrt(3**2+4**2) == 5)
>>> c
True
>>> c = math.pi*10**2
>>> c
314.1592653589793
>>> c = (2*5**2 < 7**2)
>>> c
False
```

Python program

A Python program is a sequence of Python statements

- Stored in a text file called a Python module
- Executed using an IDE or "from the command line"



```
line1 = 'Hello Python developer...'
line2 = 'Welcome to the world of Python!'
print(line1)
print(line2)
```

```
line1 = 'Hello Python developer...'
line2 = 'Welcome to the world of Python!'
print(line1)
print(line2)
```

```
$ python hello.py
Hello Python developer…
```

hello.py

Built-in function `print()`

Function `print()` prints its input argument to the IDLE window

- The argument can be any object: an integer, a float, a string, a list, …
  - Strings are printed without quotes and "to be read by people", rather than "to be interpreted by Python",
- The "string representation" of the object is printed

```
>>> print(0)
0
>>> print(0.0)
0.0
>>> print('zero')
zero
>>> print([0, 1, 'two'])
[0, 1, 'two']
```

Built-in function `input()`

Function `input()` requests and reads input from the user interactively

- It's (optional) input argument is the request message
- Typically used on the right side of an assignment statement
- The input() function will *always* treat whatever the user types as a string.

When executed:

1. The input request message is printed
2. The user enters the input
3. The *string* typed by the user is assigned to the variable on the left side of the assignment statement

```
first = input('Enter your first name: ')
last = input('Enter your last name: ')
line1 = 'Hello' + first + '' + last + '…'
print(line1)
print('Welcome to the world of Python!')
```

input.py

```
>>> name = input('Enter your name: ')
Enter your name: Michael
>>> name
'Michael'
>>> ========= RESTART =============
>>>
Enter your first name: Michael
Enter your last name: Jordan
Hello Michael Jordan...
Welcome to the world of Python!
```

Built-in function `eval()`

Function `input()` evaluates anything the user enters as a string

What if we want the user to interactively enter non-string input such as a number?

- Solution 1: Use type conversion
- Solution 2: Use function `eval()`
  - Takes a string as input and evaluates it as a Python expression

```
>>> age = input('Enter your age: ')
Enter your age: 18
>>> age
'18'
>>> int(age)
18
>>> eval('18')
18
>>> eval('age')
'18'
>>> eval('[2,3+5]')
[2, 8]
>>> eval('x')
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in
<module>
    eval('x')
  File "<string>", line 1, in
<module>
NameError: name 'x' is not defined
>>>
```

Exercise

Write a program that:

1. Requests the user's name
2. Requests the user's age
3. Computes the user's age one year from now and prints the message shown

```
>>>
Enter your name: Marie
Enter your age: 17
Marie, you will be 18 next year!
```

```
name = input('Enter your name: ')
age = int(input('Enter your age: '))
line = name + ', you will be ' + str(age+1) + ' next year!'
print(line)
```

Program Flow Control Structures

Write a program that:

1. Requests the user's name
2. Requests the user's age
3. Prints a message saying whether the user is eligible to vote or not

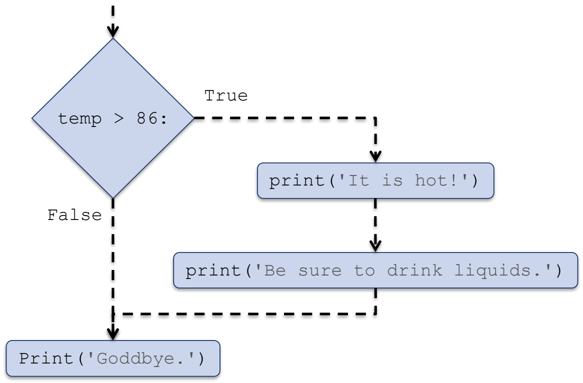
Need a way to execute a Python statement if a condition is true

One-way if statement

```
if <condition>:
    <indented code block>
<non-indented statement>
```

```
if temp > 86:
    print('It is hot!')
    print('Be sure to drink liquids.')
print('Goodbye.')
```

The value of `temp` is 90.



Exercises

Write corresponding if statements:

a) If `age` is greater than 62 then print `'You can get Social Security benefits'`

b) If string `'large bonuses'` appears in string `report` then print `'Vacation time!'`

c) If `hits` is greater than 10 and `shield` is 0 then print `"You're dead..."`

```
>>> age = 45
>>> if age > 62:
        print('You can get Social Security benefits')


>>> age = 65
>>> if age > 62:
        print('You can get Social Security benefits')


You can get Social Security benefits
>>>
```

Exercises

Write corresponding if statements:

a) If `age` is greater than 62 then print `'You can get Social Security benefits'`

b) If string `'large bonuses'` appears in string `report` then print `'Vacation time!'`

c) If `hits` is greater than 10 and `shield` is 0 then print `"You're dead..."`

```
>>> report = 'no bonuses this year'
>>> if 'large bonuses' in report:
        print('Vacation time!')


>>> report = 'large bonuses this year'
>>> if 'large bonuses' in report:
        print('Vacation time!')


Vacation time!
```

Exercises

Write corresponding if statements:

a) If `age` is greater than 62 then print `'You can get Social Security benefits'`

b) If string `'large bonuses'` appears in string `report` then print `'Vacation time!'`

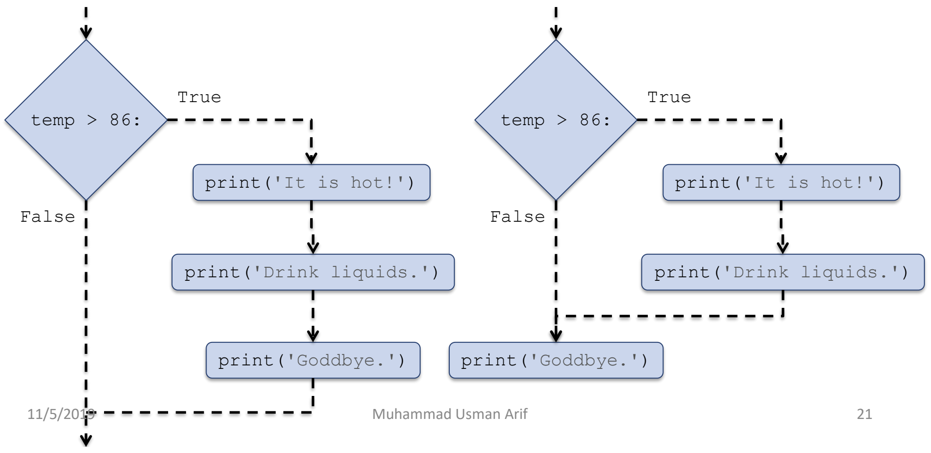
c) If `hits` is greater than 10 and `shield` is 0 then print `"You're dead..."`

```
>>> hits = 12
>>> shield = 0
>>> if hits > 10 and shield == 0:
        print("You're dead...")


You're dead...
>>> hits, shield = 12, 2
>>> if hits > 10 and shield == 0:
        print("You're dead...")


>>>
```

Indentation is critical

```
if temp > 86:
    print('It is hot!')
    print('Drink liquids.')
    print('Goodbye.')
```

```
if temp > 86:
    print('It is hot!')
    print('Drink liquids.')
print('Goodbye.')
```



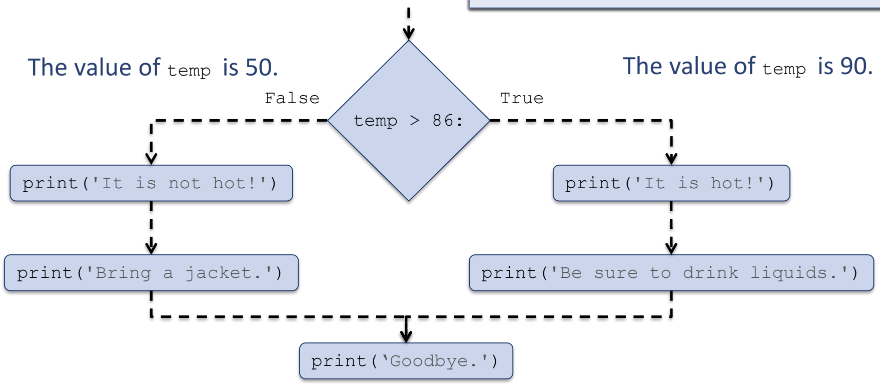
Two-way if statement

```
if <condition>:
    <indented code block 1>
else:
    <indented code block 2>
<non-indented statement>
```

```
if temp > 86:
    print('It is hot!')
    print('Be sure to drink liquids.')
else:
    print('It is not hot.')
    print('Bring a jacket.')
print('Goodbye.')
```

The value of `temp` is 50.

The value of `temp` is 90.



Exercise

Write a program that:

1) Requests the user's name
2) Requests the user's age
3) Prints a message saying whether the user is eligible to vote or not

```
name = input('Enter your name: ')
age = eval(input('Enter your age: '))
if age < 18:
    print(name + ", you can't vote.")
else:
    print(name + ", you can vote.")
```

```
>>>
Enter your name: Marie
Enter your age: 17
Marie, you can't vote.
>>>
============RESTART================
>>>
Enter your name: Marie
Enter your age: 18
Marie, you can vote.
>>>
```

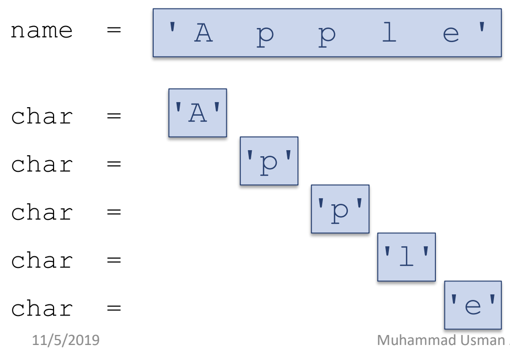
Execution control structures

- The one-way and two-way if statements are examples of execution control structures
- Execution control structures are programming language statements that control which statements are executed, i.e., the execution flow of the program
- The one-way and two-way if statements are, more specifically, conditional structures
- Iteration structures are execution control structures that enable the repetitive execution of a statement or a block of statements
- The for loop statement is an iteration structure that executes a block of code for every item of a sequence

for loop

Executes a block of code for every item of a sequence

- If sequence is a string, items are its characters (single-character strings)



```
>>> name = 'Apple'
>>> for char in name:
        print(char)


A
p
p
l
e
```

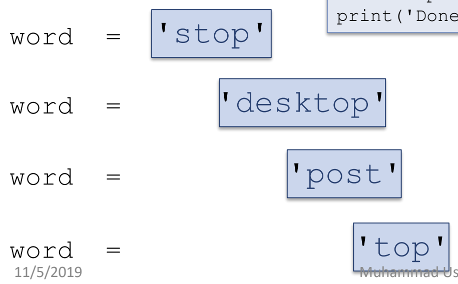
for loop

Executes a code block for every item of a sequence

- Sequence can be a string, a list, …
- Block of code must be indented

```
for <variable> in <sequence>:
    <indented code block >
<non-indented code block>
```

```
for word in ['stop', 'desktop', 'post', 'top']:
    if 'top' in word:
        print(word)
print('Done.')
```



```
>>>
stop
desktop
top
Done.
```

Exercise

Write a “spelling” program that:

1) Requests a word from the user
2) Prints the characters in the word from left to right, one per line

```
name = input('Enter a word: ')
print('The word spelled out: ')

for char in name:
    print(char)
```

```
============RESTART================
>>>
Enter a word: omnipotent
The word spelled out:
o
m
n
i
p
o
t
e
n
t
>>>
```

Built-in function `range()`

Function range() is used to iterate over a sequence of numbers in a specified range

- To iterate over the n numbers 0, 1, 2, …, n-1
  `for i in range(n):`

```
>>> for i in range(0):
        print(i)


>>>
```

```
>>> for i in range(1):
        print(i)


0
>>>
```

```
>>> for i in range(4):
        print(i)


0
1
2
3
>>>
```

Built-in function `range()`

Function range() is used to iterate over a sequence of numbers in a specified range

- To iterate over the n numbers i, i+1, i+2, …, n-1
  `for i in range(i, n):`

```
>>> for i in range(2, 6):
        print(i)


2
3
4
5
>>>
```

```
>>> for i in range(2, 2):
        print(i)


>>>
```

Built-in function `range()`

Function range() is used to iterate over a sequence of numbers in a specified range

- To iterate over the n numbers i, i+c, i+2c, i+3c, …, n-1
  `for i in range(i, n, c):`

```
>>> for i in range(2, 16, 10):
        print(i)


2
12
>>>
```

```
>>> for i in range(0, 16, 4):
        print(i)


0
4
8
12
>>>
```

```
>>> for i in range(2, 16, 4):
        print(i)


2
6
10
14
>>>
```

Exercise

Write for loops that will print the following sequences:

a) 0, 1, 2, 3, 4, 5, 6, 7, 8 , 9, 10
b) 1, 2, 3, 4, 5, 6, 7, 8, 9
c) 0, 2, 4, 6, 8
d) 1, 3, 5, 7, 9
e) 20, 30, 40, 50, 60