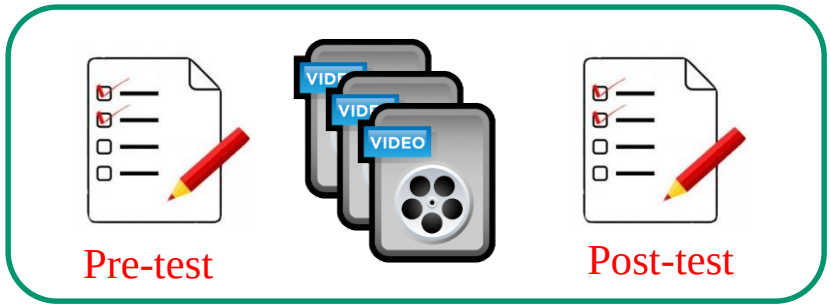
CMPUT 175
Introduction to Foundations of Computing

Crash course on Python
Quick revision of CMPUT 174 material



You should view the vignettes:

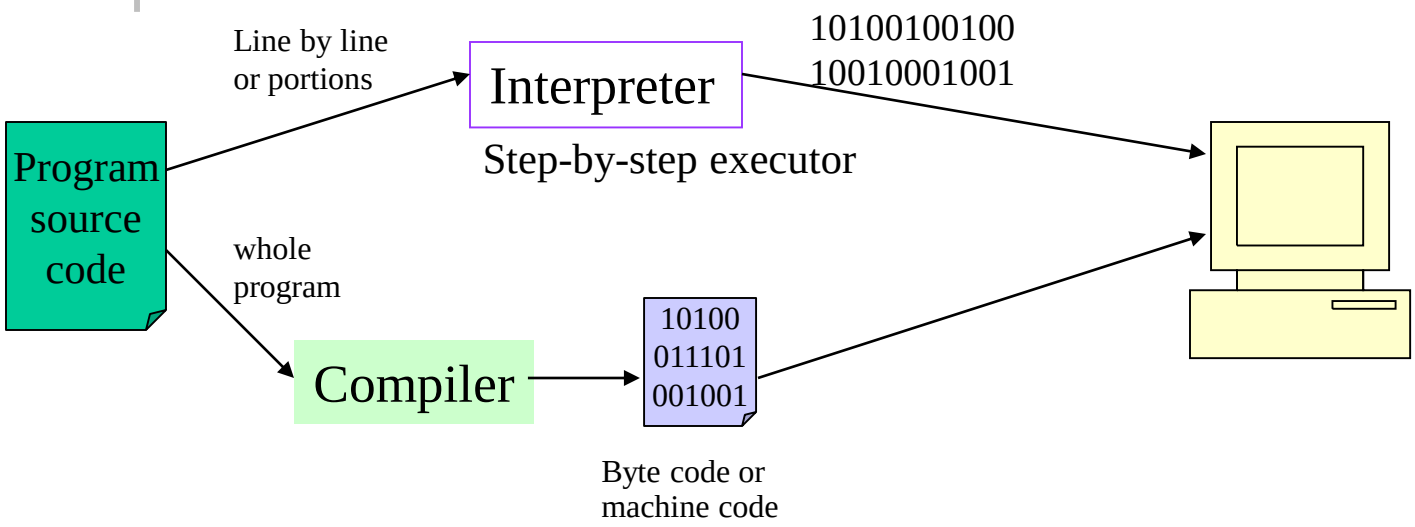
Values and types
Variables
If-statements
For-loops
While-loops
Tuples and lists
Sets
Files
Dictionaries
String Formatting

Objectives

- Revision of Programming concepts learned in CMPUT 174
- Review of Basic Python from CMPUT 174
  - What is programming
  - Variables and containers
  - Control structures
  - Functions
  - Files
  - Recursion
  - Object orientation

Programming

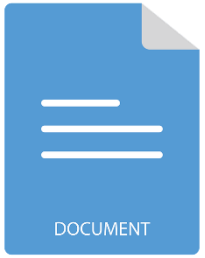
- A program is a sequence of instructions that specifies how to perform a computation. Instructions are in a programming language or machine code.
- Access data (Input); process data; provide results (Output)
- Process data: basic operations; conditional operations, repeated operations



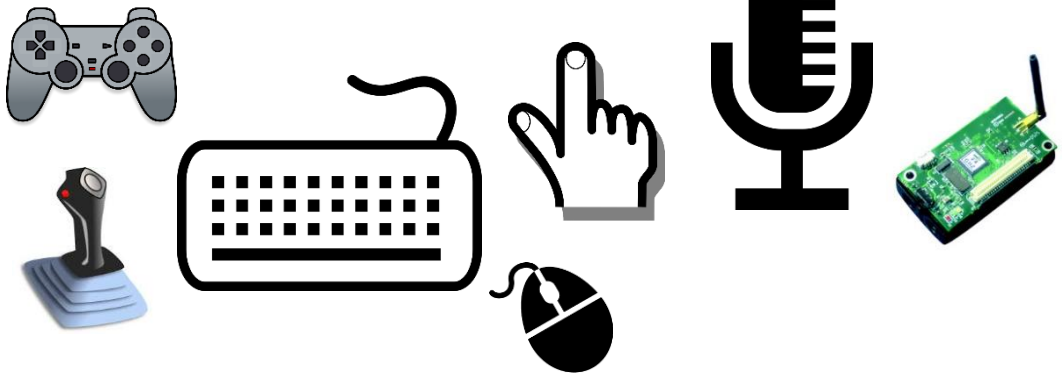
Programming

- A program is a sequence of instructions that specifies how to perform a computation. Instructions are in a programming language or machine code.
- Access data (Input); Process data; Provide results (Output)
- Process data: basic operations; conditional operations, repeated operations

variable

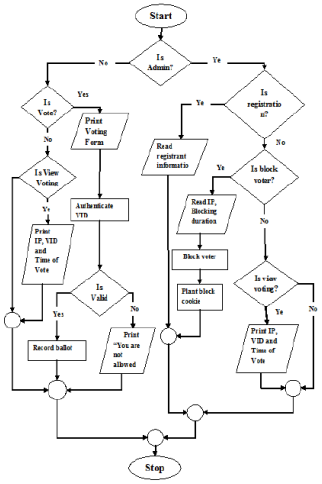


Kind of Input



Programming

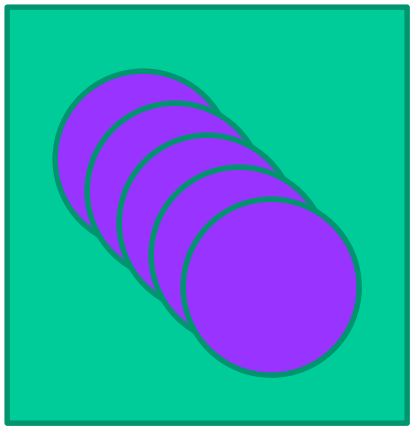
- A program is a sequence of instructions that specifies how to perform a computation. Instructions are in a programming language or machine code.
- Access data (Input); Process data; Provide results (Output)
- **Process data**: basic operations; conditional operations, repeated operations



Object Orientation

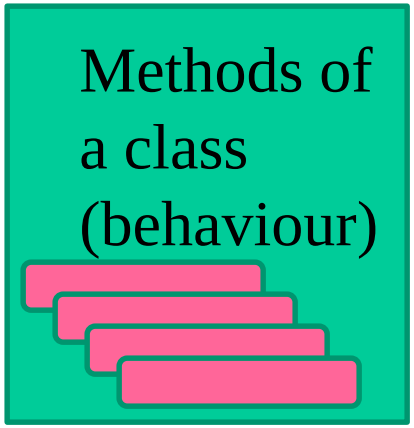
Classes

Class

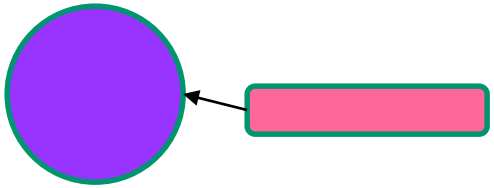


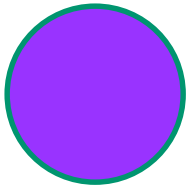
Objects

Class



Invoking a method for an Object (instance)





One Object instance from the Class

Values and Variables

- There are **values** stored in main memory
- Values have **types**:
  - String: `"this is a string"`
  - Integer: 175
  - Float: 3.1415926535897931
  - Boolean: True
  - List: `["CMPUT", 175, 91, 'A', 6]`
  - etc.
- Values can be stored, accessed and manipulated via **variables**
- Variables are names referring to values
- Python is dynamic: variables exist automatically in the first scope where they are assigned.

Variable Names

- In Python variable names should start with a letter
- Variable names can be arbitrarily long and could contain letters and numbers as well as the underscore "_"
- Python variable names are case sensitive
  - **Myvariable** is different from **myVariable**
- Python **keywords** can not be used as variable names

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

More about Variable Names

- `a = b * c` vs. `increase = salary * percent`

- Longer identifiers may be preferred: more informative and less cryptic.
- Longer identifiers add clutter making it difficult to read code.
- Shorter identifiers may be preferred as more expedient to type and perhaps easier to read code.
- Too short of an identifier makes it difficult to understand and very difficult to uniquely distinguish using automated search and replace tools.

More about Variable Names

- There are many naming convention for naming variables
- You can use multi-word identifiers
- hyphenated

```
weekly_pay = hours_worked * pay_rate
```

- Camel Case / Medial Capitals

```
weeklyPay = hoursWorked * payRate
```

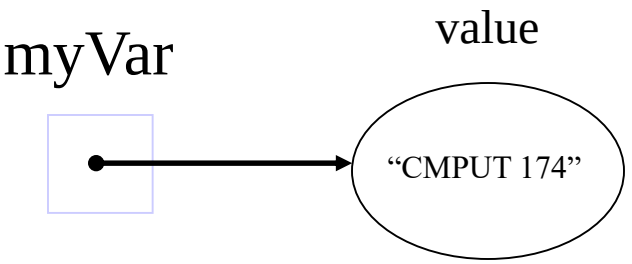
Lower camel case

```
WeeklyPay = HoursWorked * PayRate
```

Upper camel case

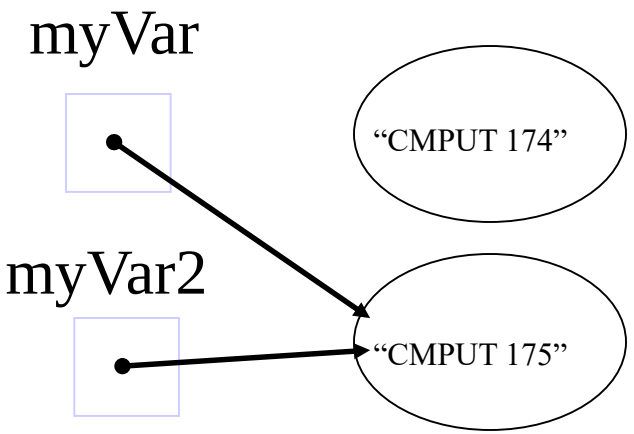
Variables are references

myVar = “CMPUT 174”



myVar = “CMPUT 175”

myVar2 = “CMPUT 175”



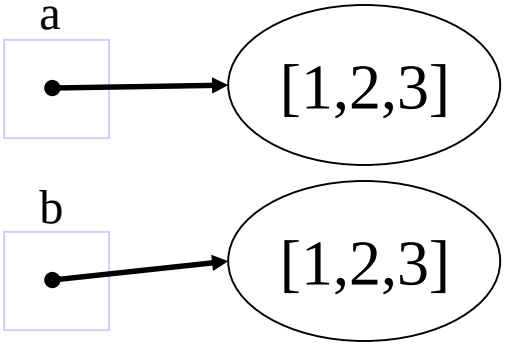
A value is assigned to a variable

What happens to the first value "CMPUT 174"?

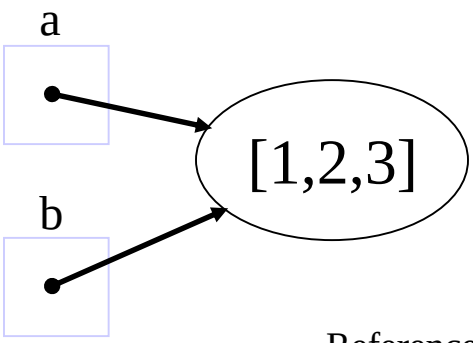
Aliasing

x=y does not make a copy of y
x=y makes x reference the same object y references

```
>>>a=[1,2,3]
>>>b=[1,2,3]
>>>a is b
False
```




```
>>>a=[1,2,3]
>>>b=a
>>>a is b
True
```

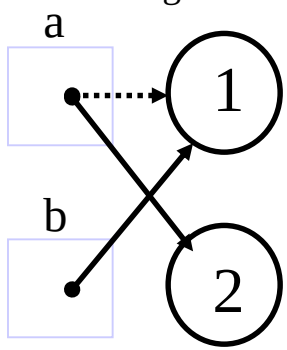


Beware

```
>>> a=[1,2,3]; b=a
>>> a.append(4);
>>> print(b)
[1,2,3,4]
```

```
>>> a=1; b=a
>>> print(b)
1
>>> a=a+1; print(b)
1
```

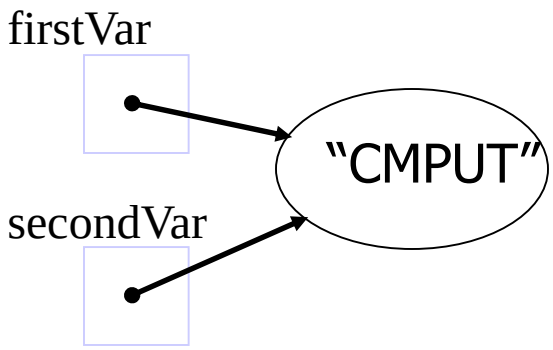
Reference lost after assignment



New object created

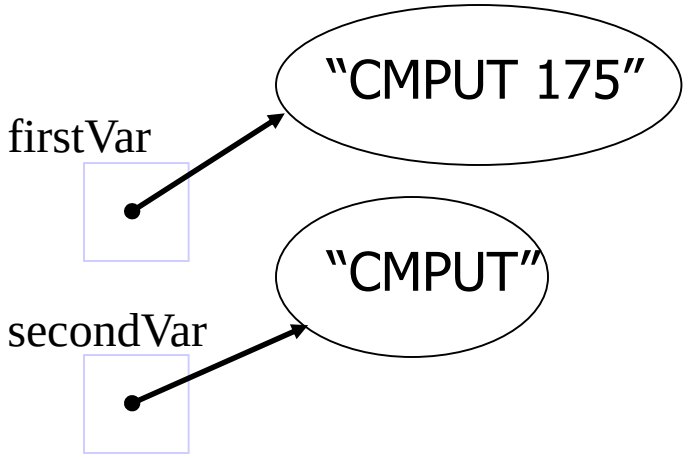
Aliasing can cause problems

firstVar= "CMPUT"
secondVar=firstVar



Use aliasing only as a second name for a mutable object.

firstVar=firstVar+ " 175"



New string object created

Types and Operators

- Integer and Float (int, float, long, complex)
  - Operators +, -, *, /, //, %, and ** perform addition, subtraction, multiplication, division, integer division, modulo and exponentiation respectively
    - 10+23 *33*
    - 67-5 *62*
    - 3*60+20 *200*
    - 210/60 *3.5*
    - 7//3 *2*
    - 7%3 *1*
    - 2**10 *1024*

Conversion functions:
abs(x) → absolute value of x
int(x) → converts x into integer
float(x) → converts x into floating point
str(x) → converts x into a string

Types and Operators

- Boolean
  - Operators and, or, not for conjunction, disjunction and negation
  - Comparison operators
    - == equality
    - != not equal
    - < less than
    - > greater than
    - <= less than or equal
    - >= greater than or equal

```
>>> False or True
True
>>> not (False or True)
False
>>> 7==2013
False
>>> 7!=2013
True
>>> (2013>=7) and (2013<=2020)
True
```

Types and Operators

- String is a sequence of characters [immutable]
  - Values of strings are written using quotations "CMPUT"
  - Characters are indexed starting from 0.
    - myVar="CMPUT"; myVar[2] returns 'P'
  - Operator + performs concatenation
    - "ABC"+"CDE" results into "ABCCDE"
  - Sequence operations
    - count(item)
    - find(item)
    - split(char)
    - lower()
    - upper()
    - center(w,char)
    - ljust(w)
    - rjust(w)
    - strip()
    - replace(old,new,max)

```
>>> myVar="CMPUT"
>>> myVar.lower()
'cmput'
>>> myVar.find('P')
2
>>> myVar.split('P')
['CM', 'UT']
>>> " CMPUT  ".strip()
'CMPUT'
```

Types and Operators

- List is a sequence of values of any type [mutable]
  - Elements in a list are numbered starting from 0
  - Accessing an element of a list via its index **k[index]**
  - Operators + and * concatenate and repeat sequences respectively
    - [1,2,3] + [4,5,6] results into [1,2,3,4,5,6]
    - [1,2,3] * 3 results into [1,2,3,1,2,3,1,2,3]
  - Operator : slices in a list
    - k=[1,2,3,4,5,6]; k[2:4] results into [3,4]
    - k=[1,2,3,4,5,6]; k[2:] results into [3,4,5,6]
    - k=[1,2,3,4,5,6]; k[:4] results into [1,2,3,4]
  - Membership operator in asks whether an item is in a list
    - 3 in [1,2,3,4,5,6] returns True
  - Length of a list with operator len
    - len([1,2,3,4,5,6]) returns 6

Methods on Lists

- append(item)
- insert(i,item)
- pop()
- pop(i)
- del(i)
- remove(item)
- sort()
- reverse()
- count(item)
- index(item)

```
L.remove(4)
print(L)
[1,3]
```

```
L.reverse()
print(L)
[3,1]
```

```
L=[1,2,3,4]
L.append(175)
[1,2,3,4,175]
```

```
L.insert(3,65)
[1,2,3,65,4,175]
```

```
L.pop()
175
```

```
L.pop(1)
2
```

```
print(L)
[1,3,65,4]
```

```
Del L[2]
print(L)
[1,3,4]
```

Lists and Strings

- list
- split
- join

```
>>> list("CMPUT")
['C', 'M', 'P', 'U', 'T']
```

```
>>> "1,2,3,,5".split(',')
['1', '2', '3', '', '5']
>>> "the cat sat on the mat".split()
['the', 'cat', 'sat', 'on', 'the', 'mat']
>>> "the,cat,sat,on,the,mat".split(',',3)
['the', 'cat', 'sat', 'on, the, mat']
```

```
>>> ' '.join(['1', '2', '3', '4', '5'])
'1 2 3 4 5'
>>> ''.join(['1', '2', '3', '4', '5'])
'12345'
>>> '**'.join(['1', '2', '3', '4', '5'])
'1**2**3**4**5'
```

Tuples and Sets

- **List** is mutable heterogeneous sequence of values
  [2, True, "cat", [1,2,3], 3.5]
- A **tuple** is an immutable list (2, True, "cat", [1,2,3], 3.5)
- Like for strings, you would get an error if you try to change the content of a tuple.
- A **set** is an unordered collection of immutable objects
  {2, True, "cat", 3.5}
- A set does not support indexing (is not sequential)
- Sets support methods such as union (|), intersection (&), issubset (<=) and difference (-), as well as add(item), remove(item), clear() and pop().

Dictionaries

- **Dictionaries** are collections of associated pairs of items
- A pair consists of a key and a value (key: value)
- Capitals={"AB":"Edmonton","BC":"Victoria","ON":"Toronto"}
- Values are accessed via their keys Capitals["AB"]
- Dictionaries are mutable
- New elements can be added Capitals["QC"]="Quebec"
- Elements in a dictionary do not have an order
- keys() returns a list of keys of dictionary
- values() returns a list of values of dictionary
- items() returns a list of pairs (key, value) of dictionary
- in returns True or False whether the key exists

BC:Victoria	AB:Edmonton	QC:Quebec	ON:Toronto

Input

- var=input("Please enter a string")
- Value entered is a string. Explicit type conversion required

- int()
- float()

```
>>> a=input('Enter number ')
Enter number 34
>>> b=a*2; print(b)
3434
>>> a=float(a);b=a*2;print(b)
68.0
```

Output

- print("CMPUT 175")

```
>>> print("CMPUT 175")
CMPUT 175
>>> print("CMPUT","175")
CMPUT 175
>>> print("CMPUT"+"175")
CMPUT175
>>> print("CMPUT","175", sep="-")
CMPUT-175
>>> print("CMPUT","175", end="-")
CMPUT 175->>>
```

- print("the class of",stdName,"is",classNumb)

String Formatting

- print(string expression % (values))
- print("%s is %d years old."% (sName,age))
  - %d or %i Integer
  - %u unsigned integer
  - %f floating point
  - %c single character
  - %s string

```
>>> user = 'username'
>>> host = 'host'
>>> '%s@%s' % (user, host)
'username@host'
```

String Formatting

- print("The rental costs $%5.2f"%(price))
- Format modifiers
  - Number %15d field width (right-justified)
  - - %-15d left-justified in field width
  - 0 %015d pre-fill with leading zeros
  - . %15.2f decimal point

```
>>> print ("%d"%(40))
40
>>> print ("%10d"%(40))
        40
>>> print ("%010d"%(40))
0000000040
```

You can also use format() function {:05.2f} instead of %05.2f

```
a=23; b=100
"My numbers are {1} and {0}".format(a, b)
My numbers are 100 and 23
```

Control structures: Conditionals

- if condition:
  statements

- if condition:
  statements
  else:
  statements

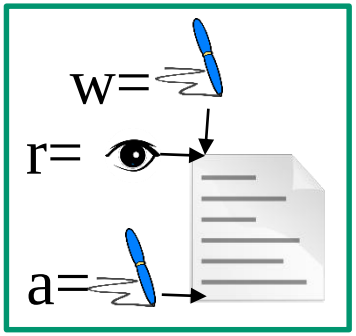
- if condition:
  statements
  elif condition:
  statements
  elif condition:
  statements …
  else:
  statements

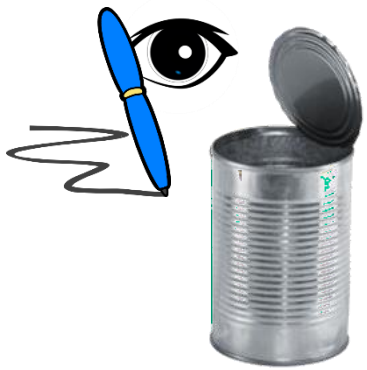
Control structures: Loops

- while condition:
  statements
- for *var* in *sequence*:
  statements
- continue
- break

Files

- f = open(filename, "r") #opens the file for reading
- f.read([byteCount]); f.readline()
- f.readlines() #returns a list where elements are lines
- f.write(string) #writes string in file
- f.close() #closes the file
- f.tell() #returns the current position in file
- f.seek([offset[,from]) #sets the file's current position at offset





- Read only ➔ "r"
- Read and write ➔ "r+"
- Write only ➔ "w"
- Append to the end of file ➔ "a"
- Append a "b" for binary file

Functions and Procedures

- def name(arg1,arg2,...):
  statements

  return # return from procedure
  return *expression* # return from function

```
>>> def sqr(x):
...     return x**2
...
>>> y= sqr(6)
>>> print (y)
36
```

A function returns a value.

```
>>> def hello(x):
...     print ("Hello %s"%(x))
...     return
...
>>> hello("CMPUT175")
Hello CMPUT175
```

A procedure executes commands.

Both a function or a procedure can be called multiple times in a program.