CCPS109 Computer Science I L1

Lecturer: Nhan Tran

n3tran@Ryerson.ca

Agenda

Announcement

Lab tomorrow:

Lecture:

variables

string

conditions

indentation

iteration

Basic Element of Python

Running python with live evaluation:

- from command line shell type `py`
- to exit use `exit() or quit()`

- Run your code inside `.py file    (mycode.py)`

```
py mycode.py
```

Basic Element of Python

Commands are statements instruct the shell to do things:

```
print(‘Hello CCPS109!’)
print(‘hello’ , ‘ world;’ )
```

Definition:

```
# this is a comment
def hello() :
    print('hello’)
    print('goodbye’)
hello()   #this run the function
```

Python’s Keywords

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Objects

- Things that python program manipulates

Two types:

- **Scalar**: indivisible, atomic
  - (four kinds): int, float, bool (True, False), Nonetype(3)

- **NonScalar**: are composite, like strings, have internal structure

Scalar Objects

- `int` – represent **integers**, ex. `9`
- `float` – **real numbers**, ex. `3.14`
- `bool` – **Boolean** values `True` and `False`
- `NoneType` – **special** -has one value, `None`

Use `type()` to see the type of an object

```
>>> type(3.0)
float
```

Type casting

Casting: covert objected of one type to another

`int(6.8)` produces 6 -truncates the number

`float(4)` convert integer 4 to 4.0

Expressions

- Form by combining objects and operators
  - Has a type and a value

```
myPi= 3+0.14
```

Operators

- `i + j`　　addition
- `i – j`　　subtraction
- `i * j`　　multiplication
- `i // j`　　floor division
- `i / j`　　division
- `i % j`　　modulus (remainder)
- `i ** j`　　exponential　or `pow(x,y)`

Operation order

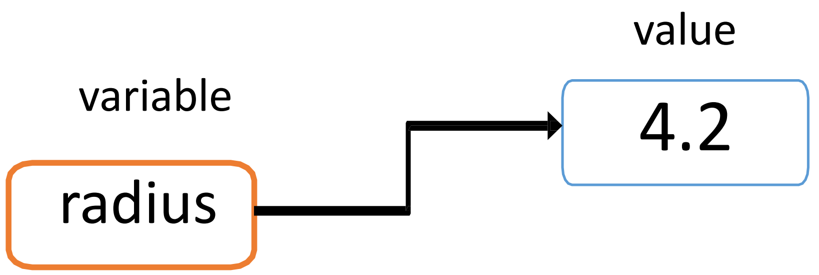
- parentheses operations first
- **operator precedence**
  - `**`
  - `*`
  - `/`

+ and – executed left to right

Variables

- Names used to assign value to
- `=` assignment binds name to value

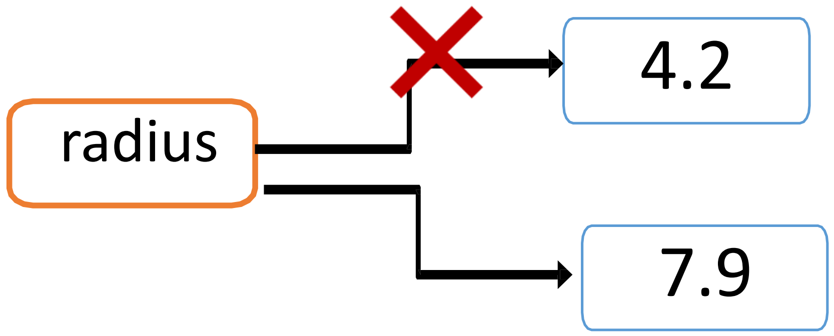
```
radius=4.2
```



Variables and associated values are store in memory while program is running.

New value may be assign to the same variable

```
radius=7.9
```



Variable names

- must start with a letter or the underscore _
- contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- cannot start with a number
- case-sensitive

  ```
  name
  Name
  NAME
  ```

  are three different variables

Abstraction

Variables allow reusability and code maintainability

```
pi = 3.141
radius = 4.2
area = pi*(radius**2)          #area is 55.40724

radius = radius+1 # new value for radius 5.2

#area value is the same
```

String

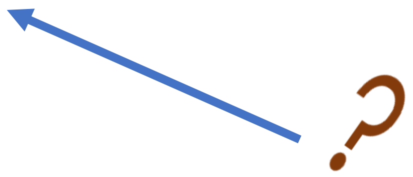
- Compose of alpha-numeric and special characters and space
- Enclose by single or double quotes

```
sayhello="hello"
name='CCPS109’
greeting=sayHello+" "+name   #concatenate
print(greeting)
```

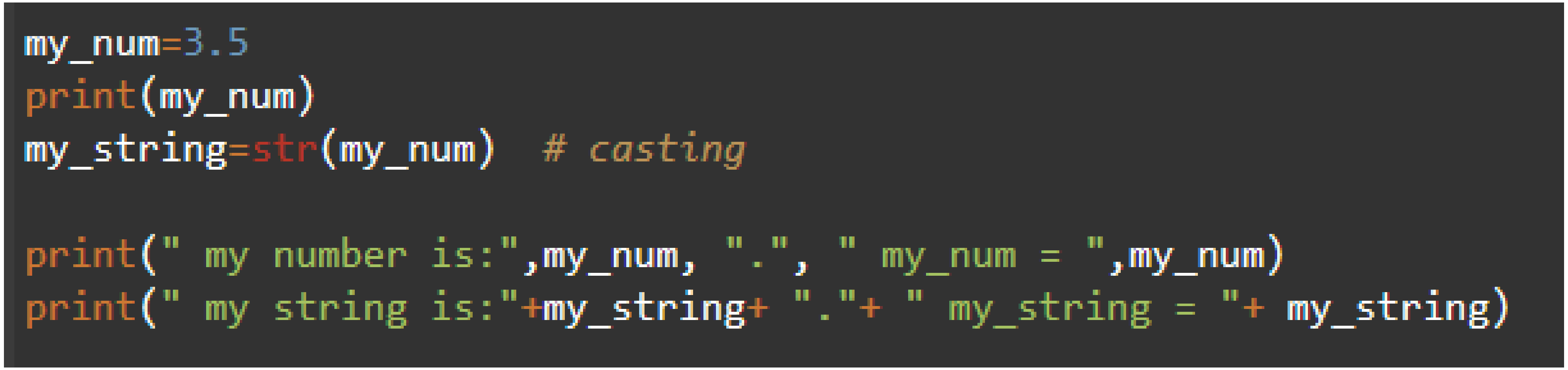
String

- Compose of alpha-numeric and special characters and space
- Enclose by single or double quotes

```
sayhello="hello"
name='CCPS109’
greeting=sayHello+" "+name    #concatenate
print(greeting)
```



String



```python
my_num=3.5
print(my_num)
my_string=str(my_num)  # casting

print(" my number is:",my_num, ".", " my_num = ",my_num)
print(" my string is:"+my_string+ "."+ " my_string = "+ my_string)
```

String: input

- User input uses `input()` method
  - The input method return a string

```
name = input("Enter name:")
print("Hello: " + name)

numbr = int(input("Enter a number:"))  #casting
print("Hello: " , numbr)
```

String: operations

- Overloaded operators
  - +   #concatenation
  - *   #repetition
- Length
  - len () method return length of string

    `len(“abcd”)`

`Indexing`

- `Individual characters or range can be extracted via indexing`

String: indexing

Indexing

- Individual characters can extracted via indexing

```
print("abcd"[0])
k='alfred is not batman’
print(k[1])

print(k[-1]) #n  negative indexing

print(k[-7:-2])  #slicing negative index
```

String methods:

```
strip()
```
removes whites space at the beginning or the end of string

```
    k='   alfred is not batman    '
    print(k.strip())
    print(k) # notice strip() did not change k
```

`lower()` return string in lower case

`upper()` return string to lower case

`title()` convert 1st letter of each word upper case

```
    k=k.title()
    print(k)            #k string is changed
```

For more String Methods:
https://docs.python.org/3/library/stdtypes.html#string-methods

String: multiline

Multiline string can be done by three, single or double, quotes

```
myString="""hello,
     Subject: CCPS109 String
     Many string operations are build into python.
     Cheers"""
print(myString)
```

String escape

Escape character use insert illegal char in python string

\"	double quote
\’	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\ooo	Octal value
\xhh	Hex value

Boolean Logic Operators

Boolean operations, ordered by ascending priority:

Operation		
x or y	if *x* is false, then *y*, else *x*	(1)
x and y	if *x* is false, then *x*, else *y*	(2)
not x	if *x* is false, then True, else False	(3)

1&2 are short-circuit operators -it only evaluates the second argument if the first one is true.

3) **not** has a lower priority than non-Boolean operators,
`not a == b` is interpreted as `not(a == b)`,
and `a == not b` is a syntax error.

Logic Operators

X	Y	X and Y	X or Y
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Comparisons

Operation		
<	less than	x < y
<=	less than or equal	x <= y
>	greater than	x > y
>=	greater than or equal	x >= y
==	Equality	x == y
!=	inequal	x != y
is	object identity (are the variables refer to the same object (more useful with nonscalar)	
is not	negated object identity	

Control Flow/Branching

```
if Boolean expression is true:
     execute this block of code
```

---

or

```
if Boolean expression:
     block of code
else:
     block of code
```

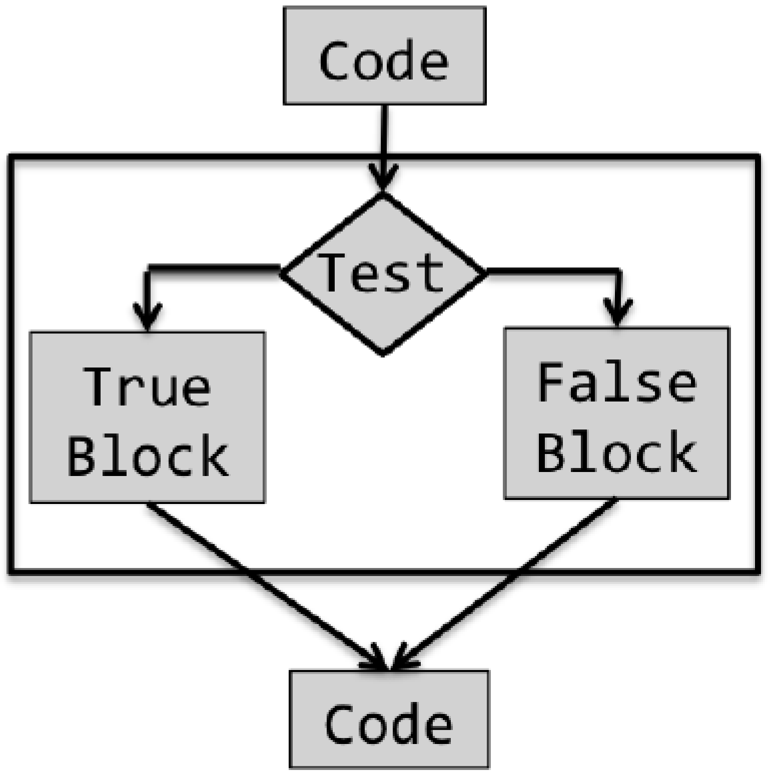


Figure 2.3 Flow chart for conditional statement

Compound Boolean expression

```
if x < y and x < z:
    print('x is least')
elif y < z:
    print('y is least')
else:
    print('z is least')
```

Control Flow/Branching

```
Or
if Boolean expression:
     block of code
elif Boolean expression:
     block of code
elif Boolean expression:
     block of code
else:
     block of code
```

More `if conditions` can be nested inside of the code block

Control Flow/Branching

```
if x%2 == 0:
     if x%3 == 0:
          print('Divisible by 2 and 3’)
     else:
          print('Divisible by 2 and not by 3')
elif x%3 == 0:
     print('Divisible by 3 and not by 2')
```

More `if conditions` can be nested inside of the code block

Indentation

- Semantically meaningful in python.
- Represent delineate block of code
- Ensure the visual structure is an accurate representation of the semantic structure.
- Typically indentation are done via tabs

Iteration (aka Loops)

- Doing the same thing many times
- Instead of writing out the same code again and again. We use iterations/ loops.
- Like conditionals: the conditional is evaluated then execute the code block(loop body).
- Once done with the loop body, recheck the condition to see if it still true.
- Jump to next code block if condition is false

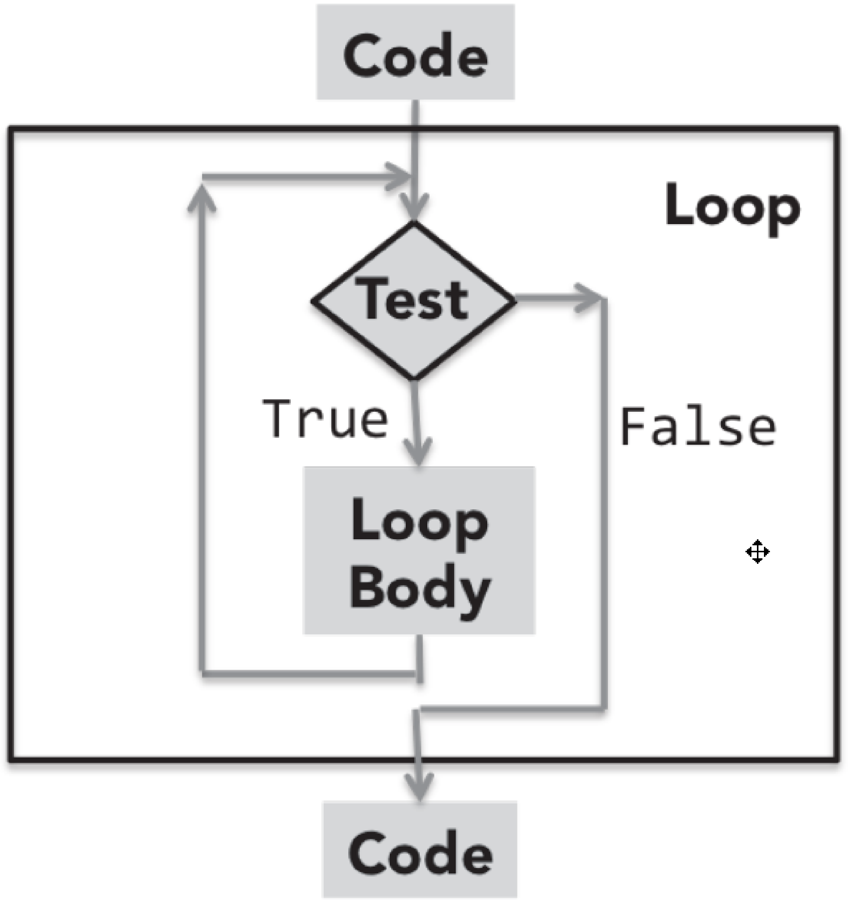


Figure 2.4 Flow chart for iteration

Iteration (aka Loops)

- While loop

```
i = 1
while i <= 12:
  print(i)
  i += 1  #i=i+1
Print(" I am outside of
while loop")
```

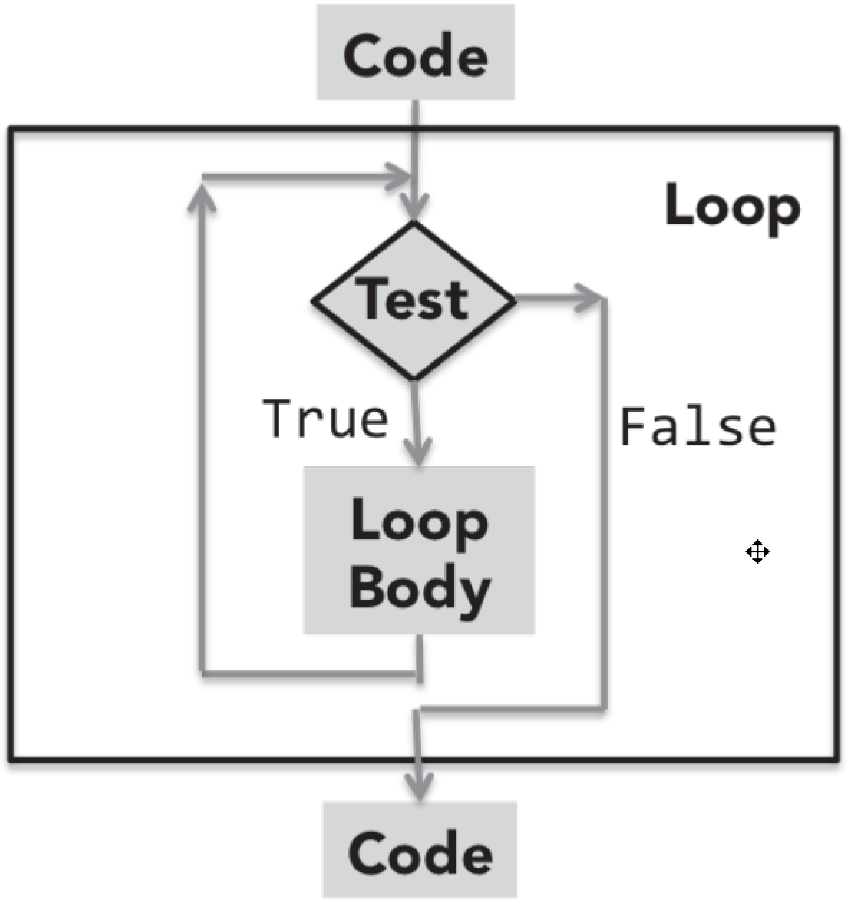


Figure 2.4 Flow chart for iteration

While

```
i = 1
while i <= 12:
  print(i)
  if i==5:
    break   #exit while loop
  i += 2
Print(“ I am outside of while loop”)
```

While

```
i = 1
while i <= 12:
  i += 2
  if i==5:
    continue   #skip the rest of loop code
                #continue to the next iteration
  print(i)
Print(“ I am outside of while loop”)
```

Look up: while loop with else

Lab hint:

```
import random
y=random.randint(0,50)
print("my random integer: ",y)
```

End