Python 101

CS*101* `lec05`

`If True: repeat in Loops`

2019-09-18

Announcements

quiz: `quiz02` and `quiz03` on Thurs 09/19

lab: `lab01` on Fri 09/20

hw: `hw02` due on Mon 09/23

hw: `hw01` last chance on Thurs 09/19

Recap on Data types and Boolean

String operations

Concatenation: combine two strings

Uses the `+` symbol

```
'RACE' + 'CAR'
```

Repetition: repeat a string

Uses the `*`

```
'HELLO '*10
```

Formatting: used to encode other data as string

Uses `%` symbol

Example

```
x = 3
s = ("%i" % (x+1)) * x**(5%x)
print(s)
```

What does this program print?

A `333333333333`

B `444444444`

C `9999`

D `%i%i%i%i%i`

Example

```
x = 3
s = ("%i" % (x+1)) * x**(5%x)
print(s)
```

What does this program print?

- A `333333333333`
- B `444444444` ⋆(Trace the steps!)
- C `9999`
- D `%i%i%i%i%i`

Example

```
x = 3
s = ("%i" % (x+1)) * x**(5%x)
```

Example

```
x = 3
s = (”%i” % (x+1)) * x**(5%x)
s = (”%i” % (3+1)) * 3**(5%3)
```

Example

```
x = 3
s = ("%i" % (x+1)) * x**(5%x)
s = ("%i" % (3+1)) * 3**(5%3)
s = ("%i" % (4)) * 3**2
```

Example

```
x = 3
s = ("%i" % (x+1)) * x**(5%x)
s = ("%i" % (3+1)) * 3**(5%3)
s = ("%i" % (4)) * 3**2
s = ("4") * 9
```

Example

```
x = 3
s = (”%i” % (x+1)) * x**(5%x)
s = (”%i” % (3+1)) * 3**(5%3)
s = (”%i” % (4)) * 3**2
s = (”4”) * 9
s = ”4” * 9
```

Example

```
x = 3
s = ("%i" % (x+1)) * x**(5%x)
s = ("%i" % (3+1)) * 3**(5%3)
s = ("%i" % (4)) * 3**2
s = ("4") * 9
s = "4" * 9
s = "444444444"
```

Example: format operator %

```
x = 666
y = ’%d’ %x
```

Example: format operator %

```
x = 666
y = '%d' %x
```

$\rightarrow$ `'666'` and `type(y) = string`

```
y = '%.1f' %x
```

Example: format operator %

```
x = 666
y = '%d' %x
```

$\rightarrow$ `'666'` and `type(y) = string`

```
y = '%.1f' %x
```

$\rightarrow$ `'666.0'` and `type(y) = string`

Boolean logic

What is the value of `x`?

```
x = True and not False or True
```

Boolean logic

What is the value of `x`?

```
x = True and not False or True
```

ans: `True`

Boolean logic

What is the value of `x`?

```
x = True and not False or True
```

ans: `True`

Order (highest priority listed first):

Boolean logic

What is the value of `x`?

```
x = True and not False or True
```

ans: `True`

Order (highest priority listed first):

```
not, and, or
```

Boolean logic

What is the value of `x`?

```
x = True and not False or True
```

ans: `True`

Order (highest priority listed first):

```
not, and, or
```

but... these operators have lower priority than:

```
 <, <=, >, >=, ==, !=
```

which have the same importance among them

Casting

What is the value of `y`?

```
y = int('32')
```

Casting

What is the value of `y`?

```
y = int('32')
```

'32' is a string, how many characters inside '32'?
How is each character represented inside a computer?

Casting

What is the value of `y`?

```
y = int('32')
```

'32' is a string, how many characters inside '32'?
How is each character represented inside a computer?

```
y = int('32') => int(051 050) => ASCII table =>
```

Casting

What is the value of `y`?

```
y = int('32')
```

'32' is a string, how many characters inside '32'?
How is each character represented inside a computer?

```
y = int('32') => int(051 050) => ASCII table =>
```

ans: `32`

```
y = int('32.0')
```

What kind of string will you expect in the ( ) for `int()`?
Will `int()` expects a '.'?

Casting

What is the value of `y`?

```
y = int('32')
```

'32' is a string, how many characters inside '32'?
How is each character represented inside a computer?

```
y = int('32') => int(051 050) => ASCII table =>
```

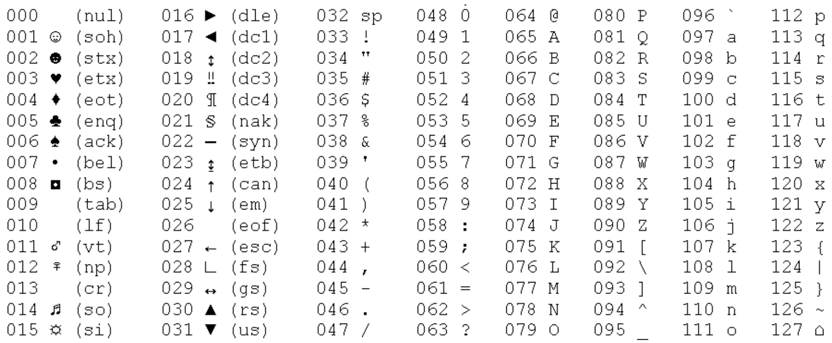
ans: `32`

```
y = int('32.0')
```

What kind of string will you expect in the ( ) for `int()`?
Will `int()` expects a '.'?

ans: `Error because int() looks up the ASCII table for representations between '0' - '9' and a few other things.....`

Casting - ASCII



000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Roadmap

Week 1 (9/9): L0 Intro; L1 PY - Intro	Week 2 (16/9): L2 PY – Data& Bool; L3 PY – if & Loops	Week 3 (23/9): L4 PY - Lists; L5 PY - Funcs	30/9: L6 PY - Files; ✗
9/10: L7 PY - Mem; 12/10: Exam 01	Week 6 (14/10): L8 PY - Dict; L9 PY - IO	Week 7 (21/10): L10 PY – Sym I; L11 PY – Sym II	Week 8 (28/10): L12 PY – Sym III; L13 PY - Numpy
Week 9 (4/11): L14 PY - Plot; L15 PY - State	Week 10 (11/11): L16 PY - Rand; Exam 02	Week 11 (18/11): L17 PY - Debug; L18 PY - Eqn	Week 12 (25/11): L19 PYA I; L20 PY - Opt II
Week 13 (2/12): L21 PY - Mod; L22 ML - Intro	Week 14 (9/12): L23 ML - IO; L24 ML - Poly	Week 15 (16/12): L25 ML - Stats; L26 ML - Fit	Week 16 (23/12): Review
Week 17 (30/12): Finals			

Recap on Data types and Boolean

Objectives

A. Write correct `if` statements.

B. Employ basic loops (`for` and `while`) to generate iterative behavior.

C. Understand the use of `range` in setting up for loops.

D. Loop aids like `break` and `continue`

Conditional Execution

Control flow

Control flow

- actual sequence of lines executed by processor.

Conditional execution

- execute (or not) a block of code based on logical comparison.

Example: if

```
ans = -5
if ans < 0:
    print( ”The number was negative.” )
```

The indented/empty space important!

if

We create an `if` statement as follows:

- the keyword `if`
- a logical comparison ('results' in `bool`)
    - a indented **block** of code;

```
if TRUE:
    executes this code block
```

if

We create an `if` statement as follows:

  the keyword `if`

  a logical comparison ('results' in `bool`)

    a indented **block** of code;

```
if TRUE:
    executes this code block
```

What happens when `False`?

Example: if

```
ans = ( -15 / 3 ) + 10
if ans < 0:
    print( ”The number was negative.” )
if ans > 0:
    print( ”The number was positive.” )
if ans == 0:
    print( ”The number was zero.” )
```

if/else statement

We create an `if`/`else` statement as follows:

- the keyword `if` (a logical comparison (results in `bool`):
  - a **block** of code (`True`)
- the keyword `else`:
  - a different **block** of code (`False`)

if ():.. else: ... 2 conditions

```
if hour < 12:
    print( ”morning” )
else:
    print( ”afternoon” )

a) hour = 11?
b) hour = 23?
```

if ():.. else: ... >2 conditions

```
day = 3

if day == 1:
    print("Monday, So happy to attend CS101!")
else:
    if day == 2:
        print("Tuesday, Exciting stuff coming")
    else:
        if day == 3:
            print("Wednesday, CS101!")
        else:
            print("Boring...")
```

if/elif/else statement

We create an `if`/`elif`/`else` statement as follows:

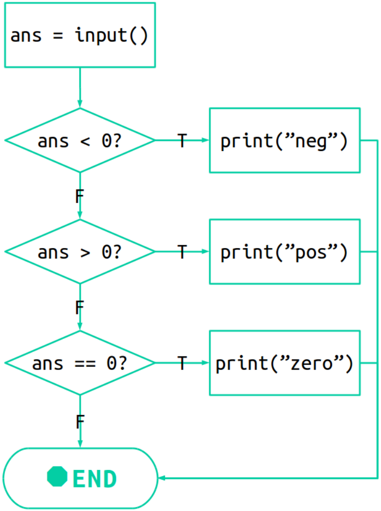
- the keyword `if` (a logical comparison (results in `bool`):
    - a **block** of code
- the keyword `elif` (a logical comparison (results in `bool`):
    - a different **block** of code
- the keyword `else`:
    - a different **block** of code

Example: if/elif/else

```
if day == 1:
    print(”Monday, So happy to attend CS101!”)
elif day == 2:
    print(”Tuesday, Exciting stuff coming”)
elif day == 3:
    print(”Wednesday, CS101!”)
else:
    print(”Boring...”)
```

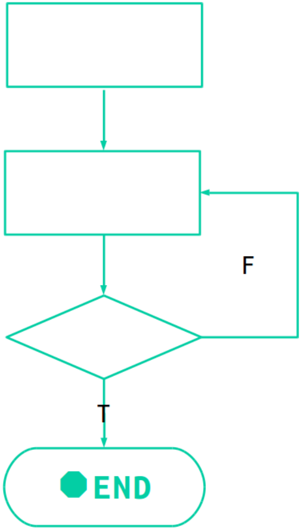
Flowchart - if

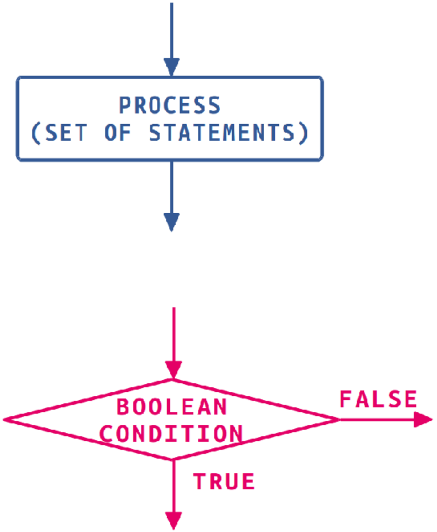
```
ans = input( "Enter a number:" )
ans = float( ans )
if ans < 0:
    print( "The number was negative." )
if ans > 0:
    print( "The number was positive." )
if ans == 0:
    print( "The number was zero." )
```



Loops

Flowchart - loop





Example: while

```
number = 10
while number > 0:
    print(number)
    number = number - 1
print('Blast off!')
```

Indentation is important!

Defining loops: while

A `while` loop has only:

- the keyword `while`
- a logical comparison (`bool`-valued result)
  - a **block** of code

```
while True:
    executes this code block
```

Example

```
ans = 'Jialing'
while ans == 'Jialing':
    ans = input( 'Jiawei and Jialing were in a
                  boat.  Jiawei fell out.
                  Who was left? ' )
```

Example

```
x = 2
while x > 0:
    print(”Hello”)
    x -= 1
```

How many times is `’Hello’` printed?

A zero

B once

C twice

D thrice

E four times

Example

```
x = 2
```

1st run:

```
while x > 0:
while 2 > 0:
```

Example

```
x = 2
```

1st run:

```
while x > 0:
while 2 > 0:

    print("Hello")      # 1st 'Hello'
```

Example

```
x = 2
```

1st run:

```
while x > 0:
while 2 > 0:
    print(”Hello”)      # 1st ’Hello’
    x -=1
    x = x - 1
    x = 2 - 1
    x = 1
```

Example

2nd run:

```
while x > 0:
while 1 > 0:
```

Example

2nd run:

```
while x > 0:
while 1 > 0:

    print("Hello")      # 2nd 'Hello'
```

Example

2nd run:

```
while x > 0:
while 1 > 0:

    print("Hello")        # 2nd 'Hello'

    x -=1
    x = x - 1
    x = 1 - 1
    x = 0
```

Example

2nd run:

```
while x > 0:
while 1 > 0:

    print(”Hello”)        # 2nd ’Hello’

    x -=1
    x = x - 1
    x = 1 - 1
    x = 0
```

3rd run:

```
while x > 0:
while 0 > 0:
```

Finished!

Example

```
x = 2
while x > 0:
    print("Hello")
    x -= 1
```

How many times is `'Hello'` printed?

- A zero
- B once
- C twice ⋆
- D thrice
- E four times

Infinite loops

Make sure that your code always has a way to end!

```
while True:
    print('Hello!')
```

Infinite loops

Make sure that your code always has a way to end!

```
while True:
    print('Hello!')
```

Use `Ctrl+C` to break free.

Accumulator pattern

Patterns are common structures we encounter in writing code.

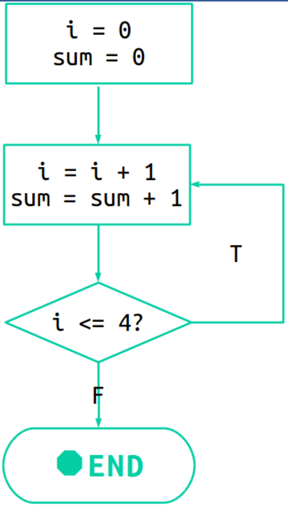
The *accumulator* pattern uses an accumulator variable to track a result inside of a loop:

```
i = 0
sum = 0
while i <= 4:
    sum += i
    i += 1
```

Note: `sum += i` is the same as `sum = sum + i`

Flowcharts

```
i = 0
sum = 0
while i <= 4:
    i = i + 1
    sum = sum + 1
```



Assuming the first test always passes (not always true)

Example

```
i = 0
sum = 0
while i <= 4:
    sum += i
    i += 1
```

What is the value of `sum`?

- A `6`
- B `10`
- C `15`
- D None of the above.

Example

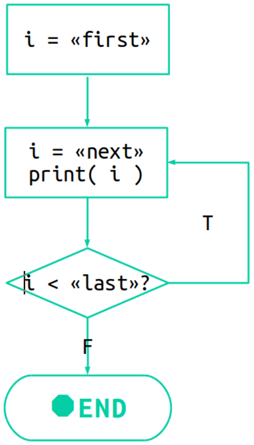
```
i = 0
sum = 0
while i <= 4:
    sum += i
    i += 1
```

What is the value of `sum`?

- A `6`
- B `10` ⋆1+2+3+4
- C `15`
- D None of the above.

Flowchart - for

```
for i in range( 0,100 ):
    print( i )
```



Defining loops: for

A `for` loop requires:

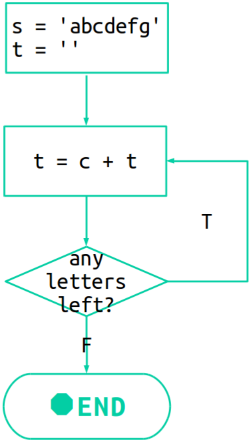
- the keyword `for`
- a loop variable e.g., `c`
- the keyword `in`
- a set of values (often `range`)
  - a **block** of code

`for` loops iterate over *iterable* types one at a time.

```
for variable in set_of_values:
    executes this code if not end of set_of_values
```

Flowchart

```
s = 'abcdefg'
t = ''
for c in s:
    t = c + t
```



Example

```
s = 'abcdefg'
t = ''
for c in s:
    t = c + t
```

What is the value of `t`?

A `'abcdefg'`

B `'gfedcba'`

C `'a'`

D `'g'`

Example

```
s = 'abcdefg'
t = ''
for c in s: => c = 'a'
```

Example

```
s = 'abcdefg'
t = ''
for c in s: => c = 'a'
    t = c + t
    t = 'a' + ''
```

Example

```
s = 'abcdefg'
t = ''
for c in s: => c = 'a'

    t = c + t
    t = 'a' + ''

for c in s: => c = 'b'
    t = c + t
    t = 'b' + 'a'
```

Example

```
for c in s: => c = 'c'
    t = c + t
    t = 'c' + 'ba'
```

.....

Example

```
for c in s: => c = 'c'
    t = c + t
    t = 'c' + 'ba'
```

.....

```
for c in s: => c = 'g'
    t = c + t
    t = 'g' + 'fedcba'
```

Example

```
for c in s: => c = 'c'
    t = c + t
    t = 'c' + 'ba'
.....
for c in s: => c = 'g'
    t = c + t
    t = 'g' + 'fedcba'
for c in s: => end of s
```

Example

```
for c in s: => c = 'c'
    t = c + t
    t = 'c' + 'ba'
```

.....

```
for c in s: => c = 'g'
    t = c + t
    t = 'g' + 'fedcba'
```

`for c in s:` => end of `s`

=> end `for` loop

Example

```
s = 'abcdefg'
t = ''
for c in s:
    t = c + t
```

What is the value of `t`?

A `'abcdefg'`

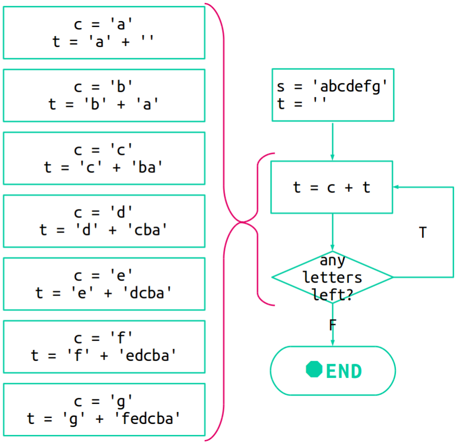
B `'gfedcba'` ⋆

C `'a'`

D `'g'`

Flowcharts

```
s = 'abcdefg'
t = ''
for c in s:
    t = c + t
```



range function

The `range` function returns a sequential set of integers.

Three arguments:

- (optional) the starting value of the range (inclusive)
- the ending value of the range (exclusive)
- (optional) the step size

Example

```
for i in range( 10,0,-1 ):
    print( i ** 2 )
```

Example

```
for i in range( 10,0,-1 ):
    print( i ** 2 )
```

```
range( 10,0,-1 ) = {10, 9, 8, 7, ... 1}
```

Example

```
for i in range( 10,0,-1 ):
    print( i ** 2 )
```

```
range( 10,0,-1 ) = {10, 9, 8, 7, ... 1}
```

```
i = 10 => print(10 ** 2) = '100'
```

Example

```
for i in range( 10,0,-1 ):
    print( i ** 2 )
```

```
range( 10,0,-1 ) = {10, 9, 8, 7, ... 1}
```

```
i = 10 => print(10 ** 2) = '100'
```

```
i = 9 => print(9 ** 2) = '81'
```

....

Example

```
for i in range( 10,0,-1 ):
    print( i ** 2 )
```

```
range( 10,0,-1 ) = {10, 9, 8, 7, ... 1}

i = 10 => print(10 ** 2) = '100'

i = 9 => print(9 ** 2) = '81'
```

....

```
i = 1 => print(1 ** 2) = '1'
```

Loop Aids

`continue`: causes the iteration to stop at where `continue` is and continue at the next index.

`break`: causes the iteration to stop at that point and also ends the loop immediately.

```
for i in range( 10 ):
    if i == 5:
        XXXX
    print( i )
```

if XXXX = `continue`, What will be printed?

Loop Aids

`continue`: causes the iteration to stop at where `continue` is and continue at the next index.

`break`: causes the iteration to stop at that point and also ends the loop immediately.

```
for i in range( 10 ):
    if i == 5:
        XXXX
    print( i )
```

if XXXX = `continue`, What will be printed?
ans: 0 1 2 3 4 6 7 8 9

if XXXX = `break`, What will be printed?

Loop Aids

`continue`: causes the iteration to stop at where `continue` is and continue at the next index.

`break`: causes the iteration to stop at that point and also ends the loop immediately.

```
for i in range( 10 ):
    if i == 5:
        XXXX
    print( i )
```

if XXXX = `continue`, What will be printed?
ans: 0 1 2 3 4 6 7 8 9

if XXXX = `break`, What will be printed?
ans: 0 1 2 3 4

Fun time 1

Write a program to sum all of the digits in a number. *i.e.*,

$$\texttt{12145} \rightarrow \texttt{1 + 2 + 1 + 4 + 5} \rightarrow \texttt{13}$$

Solution (while)

n is the number:

```
s = str( n )
i = 0
result = 0
while i < len( s ):
    result = result + int( s[i] )
    i = i + 1
```

Solution (for)

```
s = str( n )
result = 0
for x in s:
    result = result + int( x )
```

Fun time 2

Write a program which counts the number of vowels in a string. Test it on `'All ZJUI Year 1 are super-smart!'`.

Fun time 2

Write a program which counts the number of vowels in a string. Test it on `’All ZJUI Year 1 are super-smart!’`.

```
my_string = ’All ZJUI Year 1 are super-smart!’
vowel_count = 0
for char in my_string:
    if char in ’aeiou’:
        vowel_count = vowel_count + 1
```

Summary

Summary

1. `if`, `if...else:`, `if... elif:.... else:`
2. `while ....:`, `for... in.... :`
3. accumulator pattern
4. use `range( , , )`