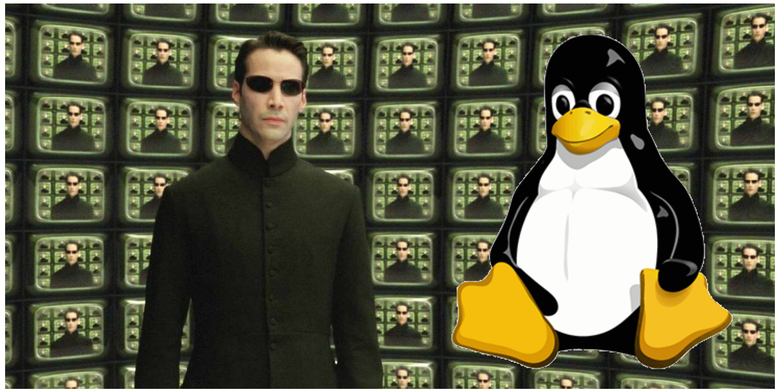
24.09.21 15:43

CSCI 120
Introduction to Computer Science and Programming I
Lecture 2: Loops II



Jetic Gū

Overview

- Focus: Basic Python Syntax
- Core Ideas:
    1. while loops
    2. More on logical expressions
    3. A bit of ASCII Code

`while` loops

Doing things, over and over again

`for` loops review

```
for VARIABLE in ITERATOR:
    SUBROUTINE...
```

- `for` loop components:
  - `ITERATOR` generates a sequence of values to `VARIABLE`
  - For each new value, the code inside the `SUBROUTINE` is executed
- Premature termination
  - `break` statement is reached: terminate the loop

Review

`while` loops

```
i = 0
while i<10:
    print(i)
    i += 1  # equivalent to i = i + 1
```

Logical expression (Condition)

Subroutine

- `while` loop
  - Infinite loop: will execute the subroutine as long as the condition is met
  - In this case, the subroutine will be executed:
    - `i` equals 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9, a total of 10 times

`while` loop

```
while CONDITION:
    SUBROUTINE...
```

- `while`'s `CONDITION` here is like the `CONDITION` you had for `if`
  - Must be convertible to `bool` type values (either `True` or `False`)
  - Arithmetic comparisons natively give out results in `bool` type

Concept

`while` loops

```
while CONDITION:
    SUBROUTINE...
```

```
for VARIABLE in ITERATOR:
    SUBROUTINE...
```

- `for` loop: uses iterator
  - Good for iterating through known values (e.g. from a `list`, from a `range()` of `int`, from `str`)
- `while` loop: uses condition
  - Good for repeated operations, keep doing until condition is no longer met

`while` loop example 1

```
while True:
    x = input()
    if x == "stop!":
        break
    else:
        print("you just typed:", x)
```

- `break` not only works for `for` loops, but also `while`
  - Good for repeated operations, keep doing until condition is no longer met

`while` loop example 2

```
i = 0
while True:
    if (i < n) is False:
        break
    # do stuff below
    i = i + 1
```

```
i = 0
while i < n:
    # do stuff below
    i = i + 1
```

- These two are equivalent
  - Correct way of comparing bool values: use the `is True` or `is False` statement

`while` loop example 3

```
while True:
    try:
        x = input()
        print("you just typed:", x)
    except EOFError:
        break
```

- `EOF` is a special character that stands for End-Of-File, to signal the end of an actual file in your computer, or the end of your input from `stdin`
  - Every file in your computer ends with `EOF`
  - To type it in console, press control-D

Logical Expressions

`and`, `or`, `not`

Python `bool` type

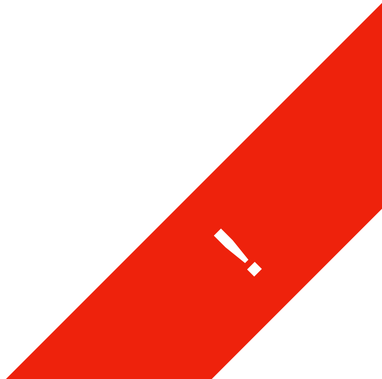
- `bool` is a special data type, with only two possible values: `True` and `False`
  - `True` and `False` are like 1 and 0, they are constants
  - You can get bool values as result of comparisons using `<`, `>`, `==`, `!=`, `<=`, `>=`

Review

The `not` operator

```
x = input()
if not (x == "Stop"):
    print("Input is not Stop")
```

- Just like in math, you can negate your comparison with logical operators
- Order of execution in the `if` statement:
  - Execute the parenthesis, compare `x == "Stop"`, get return value (e.g. `True`)
  - Execute the `not` statement, return the opposite of value of `x == "Stop"`(e.g. `False`)
- Just like in math, remember to use parenthesis so you don’t get confused with calculation order!



The `and` operator

```
x = input().split()
while True:
   if (x[0] == x[1]):
      print("it is a tie!")
   elif (x[0] == "paper" and x[1] == "rock"):
      print("paper wins!")
   ...
```

- `and`
  - The following will return `True`: `True and True`
  - The following will false `False`: `True and False`, `False and True`, `False and False`

Concept

The `or` operator

```
...
if (weather == "raining") or (weather == "snowing"):
   print("Please wear your waterproof boots!")
...
```

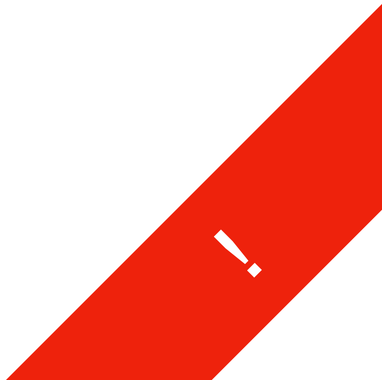
- `or`
  - The following will return `True`: `True` **`or`** `True`, `True` **`or`** `False`, `False` **`or`** `True`
  - The following will false `False`: `False` **`or`** `False`

Concept

Combine Operators

```
...
if (level == "hard") and not (name == "Batman"):
    print("You are not Batman enough to beat this level!")
...
```

- You can have multiple logical operators in the same expression
- Default order of precedence: `not` -> `and` -> `or`



Conversion to `bool`

```
while 1:
    # do stuff
```

```
x = input()
while x:
    # do stuff
```

- Certain values can be converted to bool by force (using `bool(value)`)
  - `int`: `bool(0)` -> `False`, anything else is `True`
  - `float`: `bool(0.0)` -> `False`, anything else is `True`
  - `str`: `bool("")` -> `False`, anything else is `True`
  - `list`: `bool([])` -> `False`, anything else is `True`

Concept

ASCII Code for Strings

P3
ASCII

ASCII Code

- American Standard **Code** for Information Interchange
- Your computer stores all English letters as well as basic symbols in ASCII code
- Each character in ASCII is stored as an 8bit binary number, ranging from 0-127
- For characters such as Ü, Ä, Ö, 我, 😄 (yes, include emoji), UTF8 is used, which gives indices from 128 - 2,097,152
  - Currently, we only included 1,112,064 characters, so there's room for more emoji!

Review

ASCII

'0' = 48
'A' = 65
'a' = 97
0-31: control chars

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

1. https://3.bp.blogspot.com/-EmmXIc6wNRI/WauyBOCusUI/AAAAAAAAAk8/Wodt0iDmaQowu3XuaHcIGCVes-75E7URgCLcBGAs/s1600/ASCII%2Bcode.png

P3
ASCII

Getting the ASCII code of a character

- Getting the ASCII code of a single character, use the `ord` function

  - `ord("a") -> 97`
    `ord("b") -> 98`
    `ord("c") -> 99`

  - `ord("A") -> 65`
    `ord("B") -> 66`
    `ord("C") -> 67`

  - `ord("0") -> 48`
    `ord("1") -> 49`
    `ord("2") -> 50`

  - `ord(" ") -> 32`
    `ord("\n") -> 10`
    `ord(".") -> 46`

Technical

P3
ASCII

Getting the ASCII code of a character

- Going from an ASCII code to a single character, use the `chr` function

  - `chr(97) -> "a"`
    `chr(98) -> "b"`
    `chr(99) -> "c"`

  - `chr(65) -> "A"`
    `chr(66) -> "B"`
    `chr(67) -> "C"`

  - `chr(48) -> "0"`
    `chr(49) -> "1"`
    `chr(50) -> "2"`

  - `chr(32) -> " "`
    `chr(10) -> "\n"`
    `chr(4) -> EOF`

Technical