IF STATEMENTS

Decision Structure

Conditions

- A condition is an expression
  - *Involving **relational** operators (such as < and >=)*
  - ***Logical** operators (such as and, or, and not)*
  - *Evaluates to either **True** or **False***
- Conditions used to make decisions
  - *Control loops*
  - *Choose between options*

ASCII - *American Standard Code For Information Interchange*

Ex1_ASCII.py

- ASCII values determine order used to compare strings with relational operators.
- Associated with keyboard letters, characters, numerals
  - *ASCII values are numbers ranging from 32 to 126.*
- Functions chr(n) and ord(str) access ASCII values

Standard 7-bit ASCII Table — theascii.com

Dec	Hex	Oct	Binary	Char	Description	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char
0	0	0	0	NUL	Null character	32	20	40	100000	space	64	40	100	1000000	@	96	60	140	1100000	`
1	1	1	1	SOH	Start of header	33	21	41	100001	!	65	41	101	1000001	A	97	61	141	1100001	a
2	2	2	10	STX	Start of text	34	22	42	100010	"	66	42	102	1000010	B	98	62	142	1100010	b
3	3	3	11	ETX	End of text	35	23	43	100011	#	67	43	103	1000011	C	99	63	143	1100011	c
4	4	4	100	EOT	End of transmission	36	24	44	100100	$	68	44	104	1000100	D	100	64	144	1100100	d
5	5	5	101	ENQ	Enquiry	37	25	45	100101	%	69	45	105	1000101	E	101	65	145	1100101	e
6	6	6	110	ACK	Acknowledge	38	26	46	100110	&	70	46	106	1000110	F	102	66	146	1100110	f
7	7	7	111	BEL	Bell ring	39	27	47	100111	'	71	47	107	1000111	G	103	67	147	1100111	g
8	8	10	1000	BS	Backspace	40	28	50	101000	(	72	48	110	1001000	H	104	68	150	1101000	h
9	9	11	1001	HT	Horizontal tab	41	29	51	101001	)	73	49	111	1001001	I	105	69	151	1101001	i
10	0A	12	1010	LF	Line feed	42	2A	52	101010	*	74	4A	112	1001010	J	106	6A	152	1101010	j
11	0B	13	1011	VT	Vertical tab	43	2B	53	101011	+	75	4B	113	1001011	K	107	6B	153	1101011	k
12	0C	14	1100	FF	Form feed	44	2C	54	101100	,	76	4C	114	1001100	L	108	6C	154	1101100	l
13	0D	15	1101	CR	Carriage return	45	2D	55	101101	–	77	4D	115	1001101	M	109	6D	155	1101101	m
14	0E	16	1110	SO	Shift out	46	2E	56	101110	.	78	4E	116	1001110	N	110	6E	156	1101110	n
15	0F	17	1111	SI	Shift in	47	2F	57	101111	/	79	4F	117	1001111	O	111	6F	157	1101111	o
16	10	20	10000	DLE	Data link escape	48	30	60	110000	0	80	50	120	1010000	P	112	70	160	1110000	p
17	11	21	10001	DC1	Device control 1	49	31	61	110001	1	81	51	121	1010001	Q	113	71	161	1110001	q
18	12	22	10010	DC2	Device control 2	50	32	62	110010	2	82	52	122	1010010	R	114	72	162	1110010	r
19	13	23	10011	DC3	Device control 3	51	33	63	110011	3	83	53	123	1010011	S	115	73	163	1110011	s
20	14	24	10100	DC4	Device control 4	52	34	64	110100	4	84	54	124	1010100	T	116	74	164	1110100	t
21	15	25	10101	NAK	Negative acknowledge	53	35	65	110101	5	85	55	125	1010101	U	117	75	165	1110101	u
22	16	26	10110	SYN	Synchronize	54	36	66	110110	6	86	56	126	1010110	V	118	76	166	1110110	v
23	17	27	10111	ETB	End transmission block	55	37	67	110111	7	87	57	127	1010111	W	119	77	167	1110111	w
24	18	30	11000	CAN	Cancel	56	38	70	111000	8	88	58	130	1011000	X	120	78	170	1111000	x
25	19	31	11001	EM	End of medium	57	39	71	111001	9	89	59	131	1011001	Y	121	79	171	1111001	y
26	1A	32	11010	SUB	Substitute	58	3A	72	111010	:	90	5A	132	1011010	Z	122	7A	172	1111010	z
27	1B	33	11011	ESC	Escape	59	3B	73	111011	;	91	5B	133	1011011	[	123	7B	173	1111011	{
28	1C	34	11100	FS	File separator	60	3C	74	111100	<	92	5C	134	1011100	\	124	7C	174	1111100	|
29	1D	35	11101	GS	Group separator	61	3D	75	111101	=	93	5D	135	1011101	]	125	7D	175	1111101	}
30	1E	36	11110	RS	Record separator	62	3E	76	111110	>	94	5E	136	1011110	^	126	7E	176	1111110	~
31	1F	37	11111	US	Unit separator	63	3F	77	111111	?	95	5F	137	1011111	_	127	7F	177	1111111	DEL

PE6_1

Programming Exercises – Decision Structure

1. A - O, determine the output displayed by the lines of code where *a, b, c = 2, 3, 0.* Save your code as *PE6_1.py.*

A	`print(a ** b == b ** a)`	B	`print(a < b or b < a)`
Output		Output	
C	`print('dog' > 'cat' + 'mouse')`	D	`print('Car' < 'Train')`
Output		Output	
E	`print((a == b) and ((a * a < b * b) or (b < a) and (2 * a < b)))`		
Output			
F	`print((a <= b) or ((a * a < b * b) or (b < a) and (2 * a < b)))`		
Output			
G	`print(not ((a < b) and (a < (b + a))))`		
Output			
H	`print("small" > "large" and (not c ))`		
Output			
I	`print(isinstance(c, int))`	J	`print(isinstance(3.14, float))`
Output		Output	

A	True	False
B	True	False
C	True	False
D	True	False
E	True	False
F	True	False
G	True	False
H	True	False
I	True	False
J	True	False

chr() and *ord()*

- If n is a nonnegative number, then
  chr(n)
  is the single-character string
  consisting of the character
  with ASCII value n.
- If str is any single-character string, then
  ord(str)
  is the ASCII value of the character.

A few extended ASCII values

162 ¢	177 ±	181 µ	190 ¾
169 ©	178 ²	188 ¼	247 ÷
176 °	179 ³	189 ½	248 ø

```
>>> print(chr(65))              #displays the letter 'A'
>>> print(ord('A')              #displays the number 65
```

Relational Operators (1 of 3)

Ex2_relationalOperators.py

- Relational operator can be applied to
  - *Numbers*
  - *Strings*
  - *Other objects*
- For strings, the ASCII table determines order of characters

Relational Operators (2 of 3)

Python Notation	Numeric Meaning	String Meaning
==	equal to	identical to
!=	not equal to	different from
<	less than	precedes lexicographically
>	greater than	follows lexicographically
<=	less than or equal to	precedes lexicographically or is identical to
>=	greater than or equal to	follows lexicographically or is identical to
in		substring of
not in		not a substring of

Relational Operators (3 of 3)

- Some rules
  - *An int can be compared to a float.*
  - *Relational operators can be applied to lists or tuples*

```
[3, 5] < [3, 7]
[3, 5] < [3, 5, 6]
[3, 5, 7] < [3, 7, 2]
[7, "three", 5] < [7, "two", 2]
```

Logical Operators (1 of 2)

Ex3_logicalOperators.py

- Enables combining multiple relational operators
- Logical operators are the reserved words ***and***, ***or***, and ***not***
- Conditions that use these operators are called **compound conditions**

Logical Operators (2 of 2)

- *cond1 and cond2 true only if both conditions are true*
- *cond1 or cond2 true if either or both conditions are true*
- *not cond1 is false if the condition is true, true if the condition is false*

A	B	A and B
True	**True**	**True**
False	True	False
True	False	False
False	False	False

A	B	A or B
True	True	True
False	True	True
True	False	True
False	**False**	**False**

A	not
False	True
True	False

Short-Circuit Evaluation

- Consider the condition cond1 and cond2
  - *If Python evaluates cond1 as false, it does not bother to check cond2*
- Similarly with cond1 or cond2
  - *If Python finds cond1 true, it does not bother to check further*

(a) (2 < n) and (n < 6)
(b) (2 < n) or (n == 6)
(c) not (n < 6)
(d) (answ == "Y") or (answ == "y")
(e) (answ == "Y") and (answ == "y")
(f) not (answ == "y")
(g) ((2 < n) and (n == 5 + 1)) or (answ == "No")
(h) ((n == 2) and (n == 7)) or (answ == "Y")
(i) (n == 2) and ((n == 7) or (answ == "Y"))

PE6_1

Implement the decision structure in the following:
Print "Simple"
if n is less than to m,
otherwise print "Complex".
Use the variables given below:
n, m = 7, 11

K	`if (a < b < c):`
	`    b = c + a`
	`else:`
	`    b = c * a`
	`print(b)`
		L	`if ('A' in 'apple'):`
			`    print("A as apple." )`
			`else:`
			`    print('Oops, not there.')`

Output		Output	

M	`x = 6`
	`if (x < 0):`
	`    print('negative')`
	`else:`
	`    if (x == 0):`
	`        print('zero')`
	`    else:`
	`        print('positive')`
		N	`n = 4`
			`if n <= 9:`
			`    print ("Less than ten.")`
			`elif n == 4:`
			`    print("Equal to four.")`

Output		Output	

O	`let = input("Enter A, B or C: \n")`
	`let = let.upper()`
	
	`if (let == 'A'):`
	`    print('\nA, my name is Alice.')`
	`elif (let == 'B'):`
	`    print('\nTo be, or not to be.')`
	`elif (let == 'C'):`
	`    print('\nOh, say, can you see.')`
	`else:`
	`    print('\nInvalid letter.')`		

Output			

The Boolean Data Type

Ex4_boolean.py

- Objects **True** and **False** are said to have **Boolean data type**
  - *Of data type **bool**.*
- What do these lines display?

```
x = 2
y = 3
var = x < y
print(var)
```

```
x = 5
print((3 + x) < 7)
```

Simplifying Conditions

Ex5_booleanMethods.py

- Lists or tuples can sometimes be used to simplify long compound conditions

```
(state == "MD") or (state == "VA") or (state == "WV")  or (state == "DE")
```

can be replaced with the condition

```
state in ["MD", "VA", "WV", "DE"]
```

```
(x > 10) and (x <= 20)
```

can be replaced with the condition

```
10 < x <= 20
```

```
(x <= 10) or (x > 20)
```

can be replaced with the condition

```
not(10 < x <= 20)
```

Keywords/Functions/Methods Return Boolean Values

Keywords/Functions/ Method	Returns values
in	Returns True if a value is present in a sequence (list, range, string etc.).
is	Returns True if two variables are equal
bool()	Returns the Boolean value of the specified object
Isinstance()	Returns True if a specified object is an instance of a specified object
Isdigit()	Returns True if all characters in the string are digits
startswith()	Returns true if the string starts with the specified value
endswith()	Returns true if the string ends with the specified value

https://www.w3schools.com/python/python_ref_string.asp

If Statements

If\magic_number.py & If\banned_users.py

- Also known as branching/decision structures
- Allow program to decide on course of action
  - *Based on whether a certain **condition** is true or false*
  - ***one-way** selection statement: if* statement

```
1  answer = 17
2  if answer != 42:
3      print("That is not the correct answer. Please try again!")
```

```
Python - magic_number.py:4 ✔
That is not the correct answer. Please try again!
```

```
1  banned_users = ['andrew', 'carolina', 'david']
2  user = 'marie'
3
4  if user not in banned_users:
5      print(f"{user.title()}, you can post a response if you wish.")
6
```

```
Python - banned_users.py:6 ✔
Marie, you can post a response if you wish.
```

If-else Statements

If\voting.py & If\cars.py & If\toppings.py

```
age = 18
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

Python - voting.py:7 ✔

```
You are old enough to vote!
Have you registered to vote yet?
```

- Execute the if block of statement when the condition is true
- Execute the else block of statement only when the if condition is false

```
cars = ['audi', 'bmw', 'subaru', 'toyota']

for car in cars:
    if car == 'bmw':
        print(car.upper())
    else:
        print(car.title())
```

Python - cars.py:9 ✔

```
Audi
BMW
Subaru
Toyota
```

```
available_toppings = ['mushrooms', 'olives', 'green peppers',
                      'pepperoni', 'pineapple', 'extra cheese']

requested_toppings = ['mushrooms', 'french fries', 'extra cheese']

for requested_topping in requested_toppings:
    if requested_topping in available_toppings:
        print(f"Adding {requested_topping}.")
    else:
        print(f"Sorry, we don't have {requested_topping}.")

print("\nFinished making your pizza!")
```

Python - toppings.py:1 ✔

```
Adding mushrooms.
Sorry, we don't have french fries.
Adding extra cheese.

Finished making your pizza!
```

Elif Clause

If\amusement_park.py

- An **extension** of the *if-else* statement
- Allows for more than two possible alternatives with inclusion of *elif* clauses.

```
if condition1:
    indented block of statements to execute if condition1 is true
elif condition2:
    indented block of statements to execute if condition2 is true
    AND condition1 is not true
elif condition3:
    indented block of statements to execute if condition3 is true
    AND both previous conditions are not true
else:
    indented block of statements to execute if none of the above
    conditions are true
```

```
1  age = 20
2
3  if age < 4:
4      price = 0
5  elif age < 18:
6      price = 25
7  elif age < 65:
8      price = 40
9  elif age >= 65:
10     price = 20
11
12 print(f"Your admission cost is ${price}.")
```

```
Python - amusement_park.py:7 ✔

Your admission cost is $40.
```

Nested *If-else* Statements

- Indented blocks of *if-else* and *if* statements can contain other *if-else* and *if* statements
  - *The if-else statements are said to be* ***nested***

```
1  x = 6
2  if (x < 0):
3      print('negative')
4  else:
5      if (x == 0):
6          print('zero')
7      else:
8          print('positive')
```

PE6_2 & PE6_3 & PE6_4 & PE6_5

- Write your codes and run

Summary (1 of 2)

- The word *if*, *else* and *elif* are reserved words.
- A **condition** is an expression involving literals, variables, functions and operators (arithmetic, relational, or logical) that can be evaluated as **True** or **False**.
- The relational operators are <, >, ==, !=, <=, >=, *in*, and *not in*.
- The principal logical operators are *and*, *or* , and *not*.
- The use of indentation to mark blocks of code helps Python code more readable.
- Statements consisting of a header followed by an indented block of code are called **compound statements**.
- The **Boolean data type** (or **bool**) has the two values True and False.
- When an **empty** string, list, or tuple is used as a condition, it evaluates to **False**.

Summary (2 of 2)

- The **ASCII** table associates characters with nonnegative numbers.
- The value of *chr*(n) is the character associated with the number n. *chr*(65) is ‘A’.
- The function of *ord*() is the reverse of the *chr*() function.
- The use of ASCII table to order items of data are called **lexicographical ordering**.
- The list *sort* method lexicographically orders item.
- The *in* operator can be used to simplify complex conditions.

Decision Structure - Terminologies

- ❑ 1 and
- ❑ 2 or
- ❑ 3 not
- ❑ 4 in / not in
- ❑ 5 is / is not
- ❑ 6 bool()
- ❑ 7 chr()
- ❑ 8 ord()
- ❑ 9 Isinstance()
- ❑ 10 Isdigit()
- ❑ 11 if
- ❑ 12 if-else
- ❑ 13 elif Clause
- ❑ 14 Nested if-else
- ❑ 15 True
- ❑ 16 False
- ❑ 17 ==
- ❑ 18 !=
- ❑ 19 <, <=
- ❑ 20 >, >=
- ❑ 21 ASCII
- ❑ 22 Boolean
- ❑ 23 Branching/Decision
- ❑ 24 Conditions
- ❑ 25 Compound Conditions
- ❑ 26 Indentation
- ❑ 27 Lexicographical Order
- ❑ 28 Logical Operators
- ❑ 29 Relational Operators
- ❑ 30 Short-Circuit Evaluation

Quiz 6

- Quiz 6A has 10 questions in 15 minutes, 10 pts
  - *10 multiple choice/true or false questions, 1 pt. for each question*
  - *Quiz 6A has two attempt, the higher grade will be selected*
  - *Submit Quiz 6A (at least 1-minute) **before** the due time to Blackboard*

- Quiz 6B has 2 code questions, 15 pts
  - *Write the Python code based on the given question*
  - *Each question will be given during the last 10-minute of each session of week 6*
  - *Quiz 6B-1 on session A, and Quiz 6B-2 on session B*
  - *Quiz 6B has one attempt*

DB 6

- Instruction:
  1) Choose any **two** of the questions from **PE6_1 and** any **one** of the questions from PE6_6. Please **avoid** selecting the exact same questions. Make sure to indicate the **question #** you're working on in the thread title as soon as you open your thread. Then you can **explain and edit your questions** (1.2 pt).
  2) Explain the following (0.3 pt).
  How can **short-circuit evaluation** improve performance?
  3) Submit your posts before the due date. Let's learn from each other.