CST0006 – Computer Programming Foundations

INTRODUCTION to WEEK06

In this lecture and lab section we are going to talk about begin talking about the flow of control of a program. For example, if we ask a user to login, did they get the correct login information? If not, we better tell them to try again instead of giving them access to the bank account!

We will cover conditional statements, comparison operators and Boolean logic.

Boolean Comparison

There will be many times where you will want to compare two variables to see at which state they currently exist. As we will see later, this will allow us to make decisions based on these two Boolean states.

Comparison Operators

==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Comparing Two Integers

Example:

```
a = 1
b = 2

print(a == b)
```

What would the output be?

Example:

```
a = 1
b = 2

print(a < b)
```

What would the output be?

Comparing Integers and Strings

Example:

```
a = 1
b = '1'

print(a < b)
```

What would the output be?

Comparing Two Strings

Example:

```
a = "dog"
b = "Dog"

print(a < b)
```

What would the output be?

Strings vs Characters

To understand what we just compared, we need to dive a little deeper into the logic of what just happened. Most programming languages have a data type called a “char” short for character. A string is an array of characters. Python doesn’t have a char as a built in data type. Instead, it will be just a string with the length of one.

Now to understand why “dog” is greater than “Dog”, we need to look at the ASCII Table. ASCII stands for American Standard Code for Information Interchange. Each character on the keyboard has a numeric value.

Luckily Python has some built in functions that allows us to convert ASCII characters to their decimal value and back again. So now we can see what happened by breaking down the characters into their decimal value.

`ord(i)` - converts an ASCII character into its decimal value
`chr(i)` - converts a decimal value into its ASCII character

Example:

```
a = 'd'

print(ord(a) == 100)
print(chr(100) == a)
```

What would the output be?

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

Conditional Branching

Conditional statements allow us to control the flow of our programs based on whether or not something is true or false. The flow of control based on a condition is called Conditional Branching. This allows us to do one thing if the condition is true or do another if it is false.

For example an egg timer. You will check to see if the time has expired, (True or False). If the egg timer has expired your egg is cooked, if the time hasn't expired your egg is still uncooked. Now we can decide what we want to do with it. If the condition is true, you can enjoy your egg, if the condition is false, you get salmonella poisoning and end up in the hospital.

This logic can be any of the two states that we can think in.

Did the car crash? Yes/No?

Do we still have lives left? True/False?

Which position is the light switch in? On/Off?

Which binary state are we in? 1/0?

There are tons of different scenarios we can invent to allow us to make decisions. As long as there is some variable that we can test, we can write code to make a solution based on the state we are testing.

If Statements

We do conditional branching by using “if statements”. If statements are present in most programming languages.

Here is an example of what they look like:

Python Coding Example:

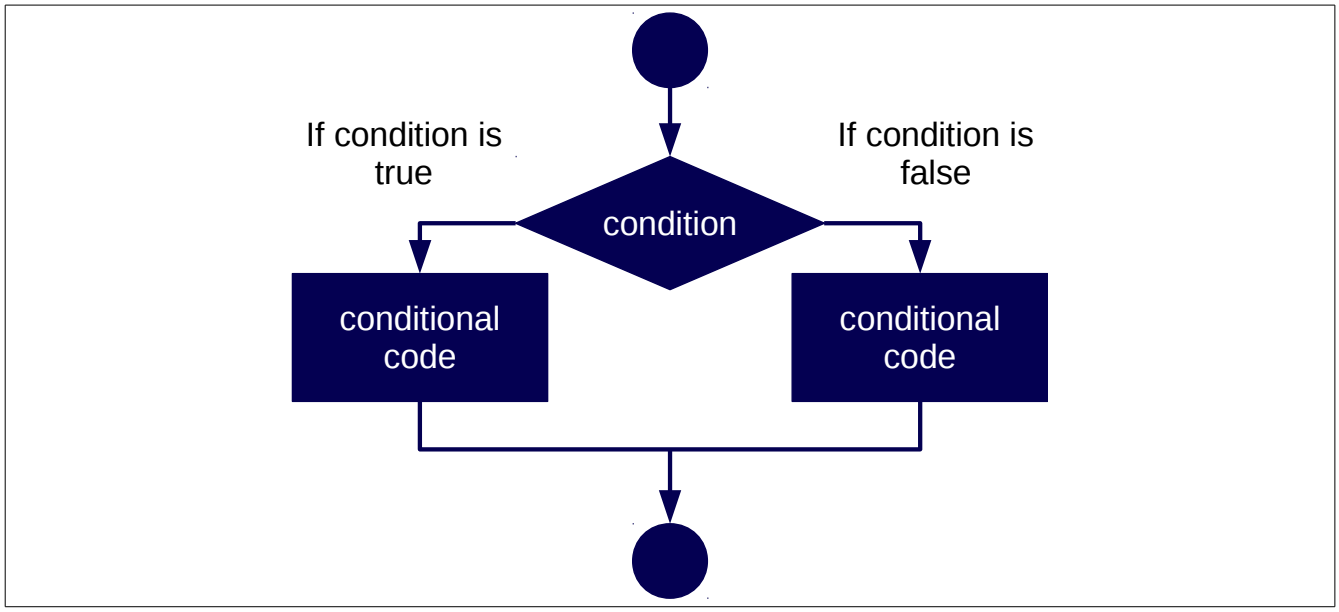
```
if condition evaluates to True:
    #then do things that you should only do if you are expecting a True
else:
    #then do things that you should only do if you are expecting a Not True
```

C Coding Example:

```
if(condition evaluates to True)
{
    /* then do things that you should only do if you are expecting a True; */
}

else
{
    /* then do things that you should only do if you are expecting a Not True; */
}
```

Flowchart Example:



You don’t always have to include and else statement. There are also times you may just want to check if something is true and act on it.

Complex Conditions - Compound if Statements

There may be some situations where you will want to check two conditions at once and if both are true, then run the code. Some languages use “&&” to represent “and” and “||” to represent “or”.

And Example:

```
num1 = 1
num2 = 2

if num1 == 1 and num2 == 2:
    print("num1 and num2 are True")

else:
    print("Everything else is not true")
```

There may be some situations where you will want to check two conditions and if any one of them are true, you can run the code.

Or Example:

```
num1 = 1
num2 = 2

if num1 == 1 or num2 == 2:
    print("num1 and num2 are True")

else:
    print("Everything else is not true")
```

Complex Conditions - Nested If Statements

There may be some situations where you need to test a condition and if that condition is true, then you want to test something else to see if it’s true.

Example:

```
num1 = 1
num2 = 2
num3 = 3

if num1 == 1:
    print("num1 is true")
    if  num2 == 2:
        print("num2 is also true")
        if num3 == 3:
            print("num3 is also true")

else:
    print("Everything else is not true")
```

Complex Conditions - True False and Other

It is important to understand that sometimes logic isn't only presented as a Boolean of true or false, 0 or 1… sometimes there's a third option to consider. Understanding this is when you start to think like a programmer.

For example:

```
num = 2

if num == 1:
    print("True")

else:
    print("False")
```

If we were expecting 1 to be True and 0 to be false, we have asked the wrong question. In this code snippet, we are saying If one is true, then everything else that isn't a one is false.

What we should be saying is:

```
num = 2

if num == 1:
    print("True")

elif num == 0:
    print("False")

else:
    print("Error: Something is wrong with the logic")
```

It's important to think like a programmer when using if statements. It is very easy to get your logic wrong when adding complexity to your if statements. There may be times that more than one condition may be true and you could execute code that you didn't plan on running. That's why using "else if" statements are important. Only one of the previous lines code blocks will run, once one of those conditions is true, then the rest of the statements are ignored.

Another danger to watch out for:

```
timer = 0

while True:
    if timer == 100:
        #do something important

    #do something else for a while
```

What if you check the timer and it's 99, you go through the loop and the timer is at 101? That important code you wanted to run will never happen. So it's good to check if it's is greater than 100.