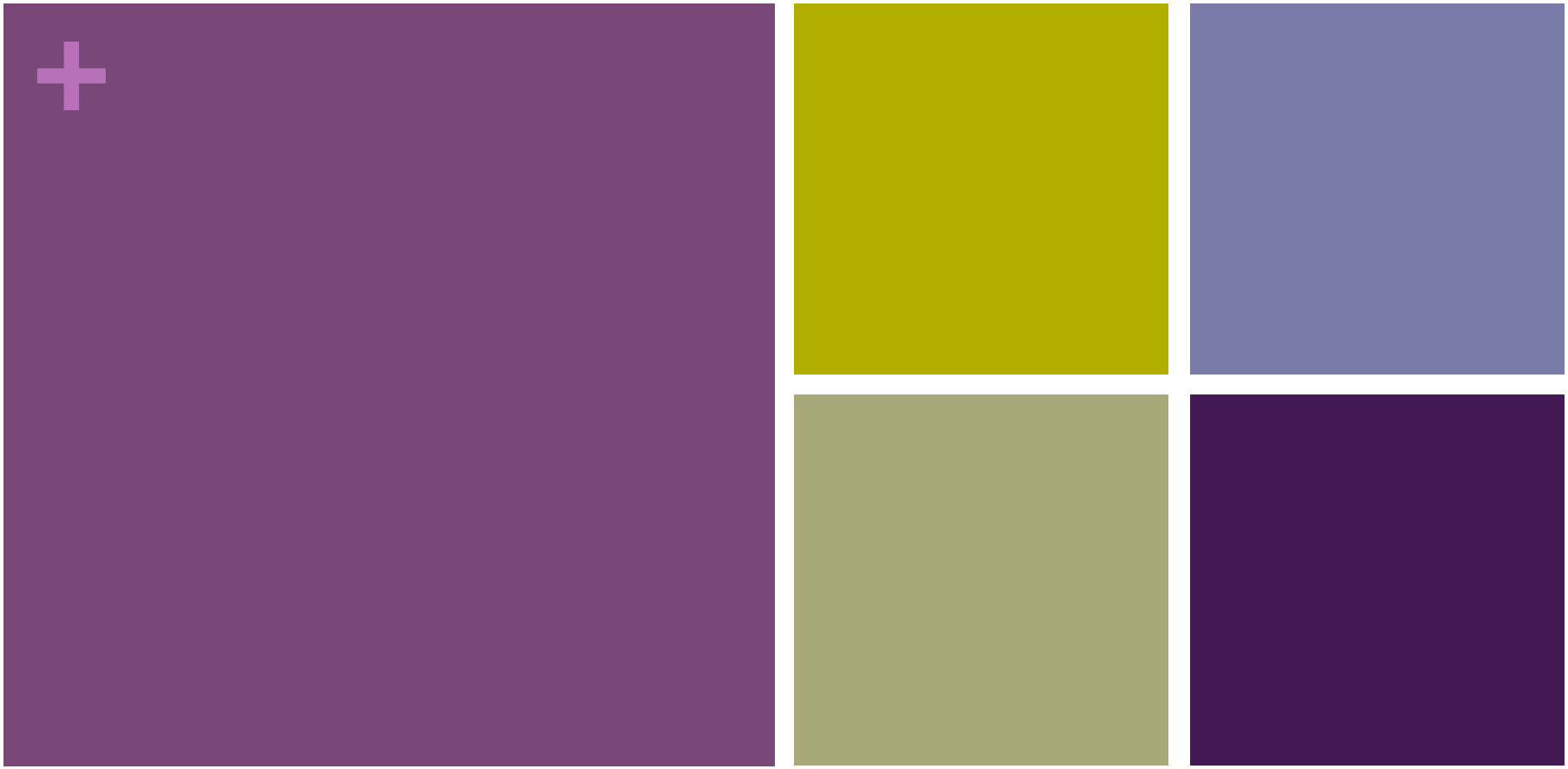
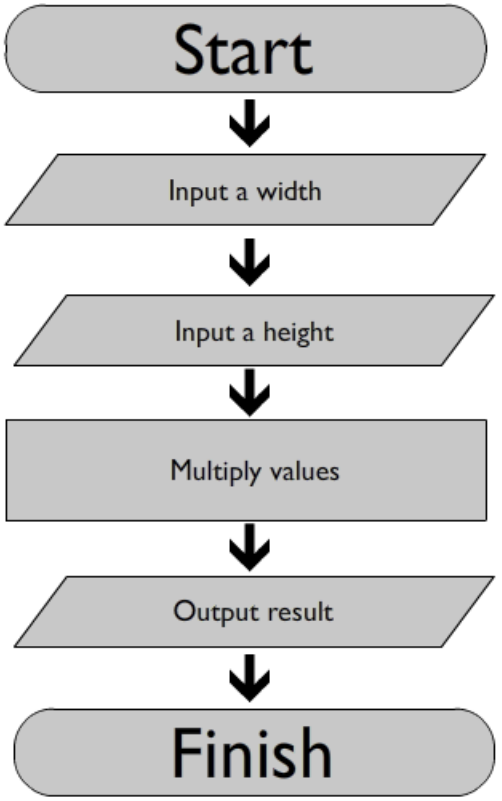


Decision Structures & Boolean Logic

CSCI-UA.002

Sequence Structures

- What we have been programming so far is known as a "sequence structure"
- Sequence structures are sets of statements that execute in the order in which they appear
- Unfortunately not all programs can be written this way, as there are certain times when we need to deviate from a linear structure and adapt our program based on information provided.



Example: Calculating Overtime Pay

- If a worker works more than 40 hours in a week he or she is entitled to overtime pay.
- Overtime pay is calculated at the rate of 1.5 times the worker's hourly rate.
- This additional rate is only applied to hours worked above the 40 hour limit.

Example: Calculating Overtime Pay

- Input: Hourly rate of pay
- Input: Number of hours worked in 1 week
- Process: If the hours worked is less than 40, simply multiply hourly rate by hours worked
- Process: If the hours worked is greater than 40:
  - Multiply hourly rate by hours worked for 40 hours.
  - Subtract 40 from the the total hours to obtain the overtime hours
  - Multiply overtime hours by 1.5 times the rate of pay
  - Add overtime pay to base pay
- Output: Total Pay

Example: Calculating Overtime Pay

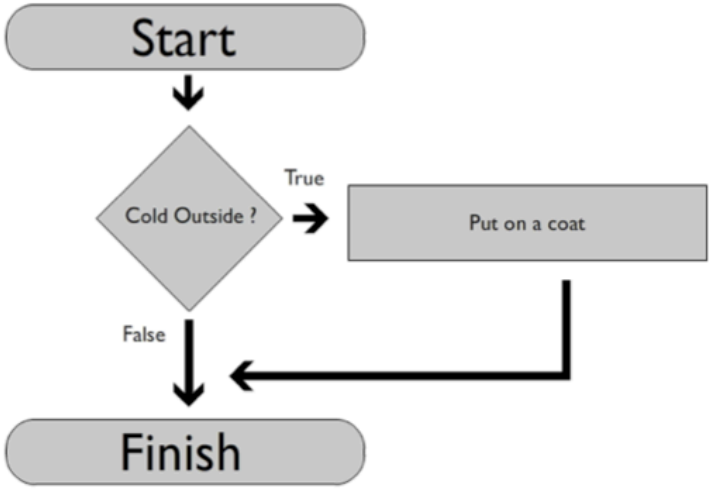
- Our current Python toolset doesn't give us the ability to deviate from a linear sequence structure

The Selection Statement

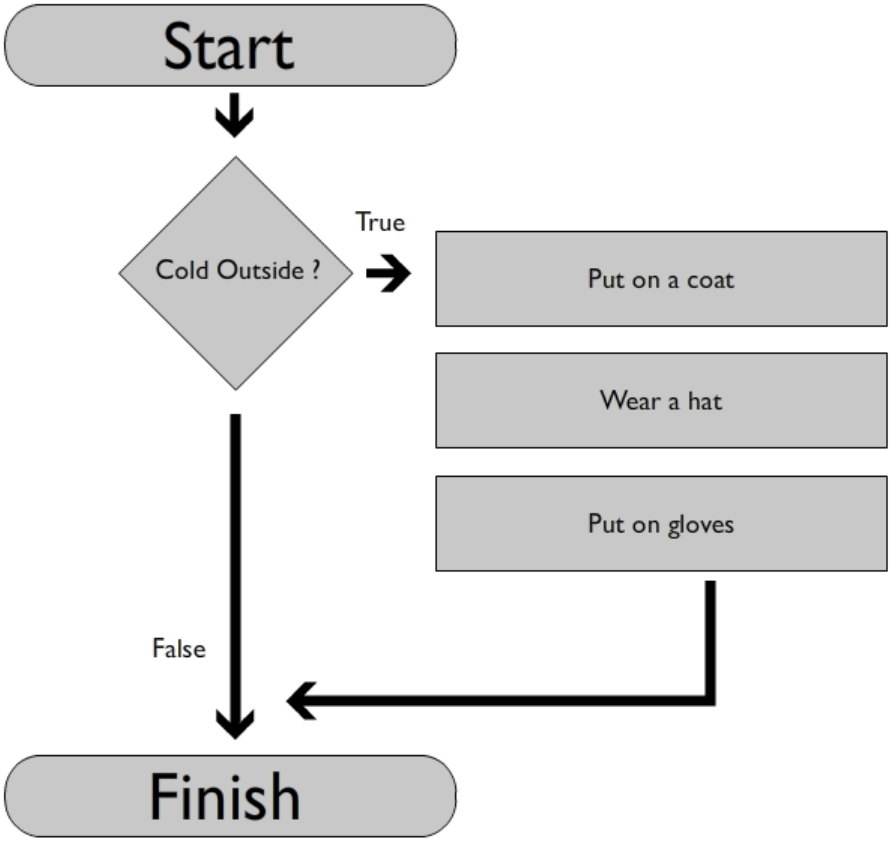
- Allows your program to "ask a question" and respond accordingly.
- Simplest form – perform an action only if a certain condition exists
- If the condition is not met, then the action is not performed

The Selection Statement

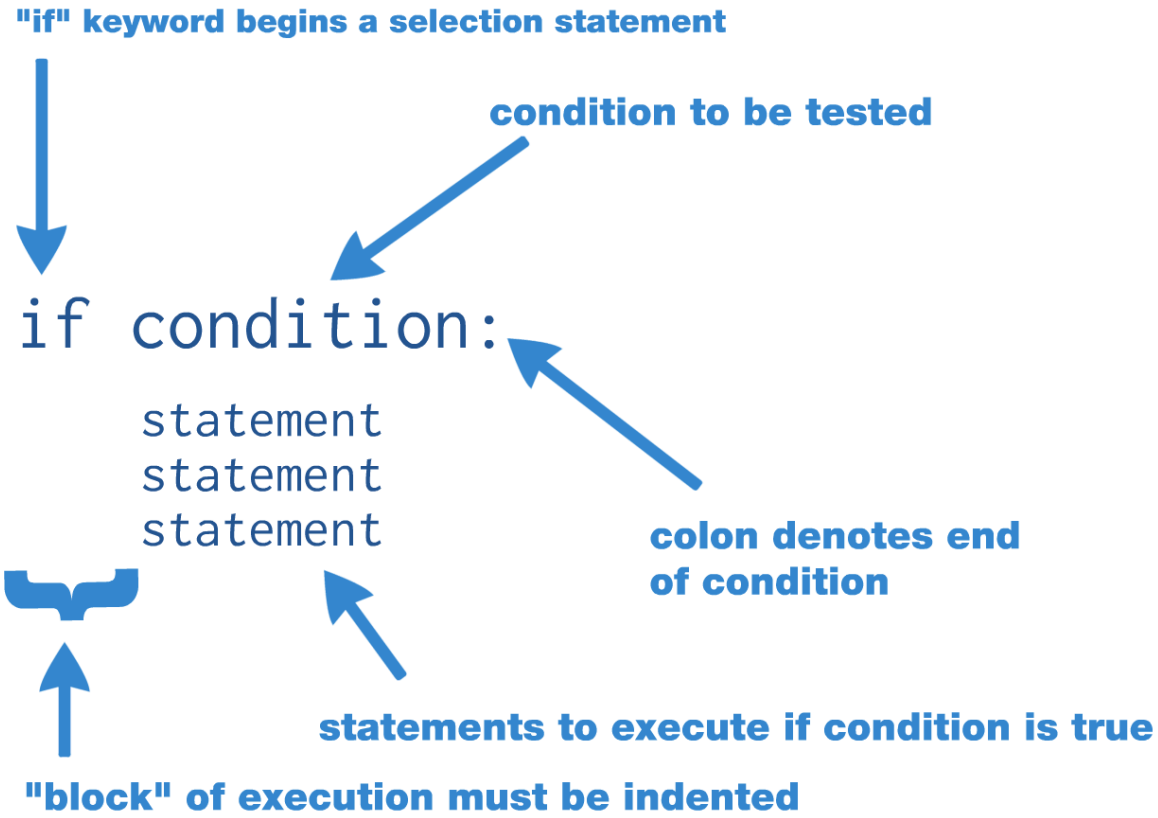
- In this program we begin by asking a question – "is it cold outside?"
- If the answer to this question is yes (aka "True") then we can execute an alternate set of commands
- Otherwise we can continue with the program as-is



The Selection Statement



Selection Statements in Python



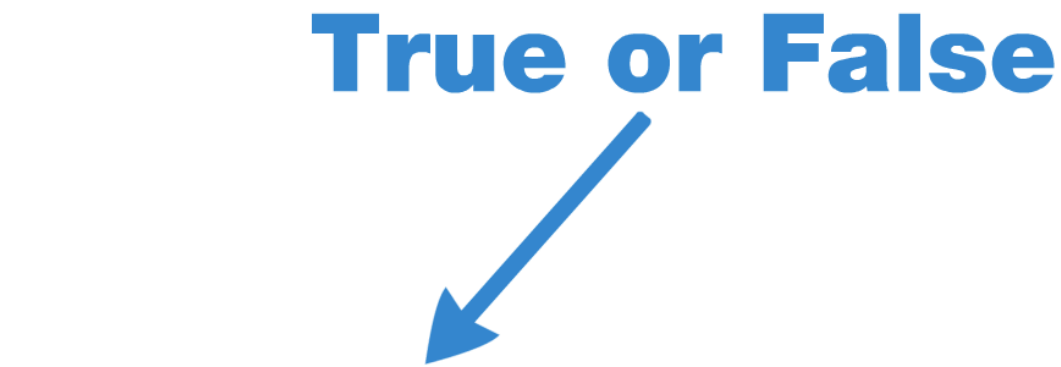
+

Boolean Expressions

Writing a condition

- The trick to writing a selection statement is in constructing a condition that matches the question you are trying to ask the computer
- All selection statements must have a condition to "test"
- Think of conditions as "yes or no" questions. They can only be answered by one of two options – "True" or "False"

Boolean Expressions




```
if condition:
    statement
    statement
    statement
```

Boolean Expressions

- Named after George Boole, a 19th century English philosopher and mathematician
- Boole developed a system of mathematics that allows us to work with the abstract concepts of "true" and "false"
- Boole is considered one of the founders of modern computer science, as his work underpins the way in which modern computers process binary data

Writing a Boolean Expression

- Boolean expressions can be used as the condition in an "if" statement
- They are generally formed using "relational operators" which allow you to test to see whether a specific relationship exists between two (or more) values

Relational Operators

```
a > b       # is a greater than b ?

a < b       # is a less than b ?

a == b      # is a equal to b ?

a <= b      # is a less than OR
            # equal to b ?

a >= b      # is a greater than OR
            # equal to b ?
```

Writing a Boolean Expression

- ALL Boolean expressions boil down to “True” or “False”
- Programmers often say that the expression “evaluates” to “True” or “False”

Writing a Boolean Expression

```
pen = 10

sword = 7


if pen > sword:                  # pen > sword

  print ('the pen is            # 10 > 7
  mightier than the
  sword!')                       # True
```

Let's Evaluate!

```
# given these variables

a = 99

b = 7

c = -5

d = 92
```

```
# evaluate these expressions

a > b

b < c

b >= c

c <= d

a == b + d

d <= a + c

c != b
```

Boolean Operator Tips

- Don't confuse "==" with "="
  - "=" is used for assigning values to variables
  - "==" is used for testing to see if two values are identical
- Use "!=" if you want to test if two values are different
- The "<=" and ">=" operators test for more than one relationship
  - "<=" tests to see if a value is less than OR equal to another
  - ">=" tests to see if a value is greater than OR equal to another

+

Let's write some programs!

Programming Challenge: Freezing / Boiling Guppies

- Guppies are hardy fish, but they can't live in all water temperatures.
- The acceptable range for guppies is between 72 and 86 degrees Fahrenheit.
- Write a program that asks the user for a temperature. Then display one of two messages based on the information provided:
  - You're going to freeze your guppy!
  - You're going to boil your guppy!

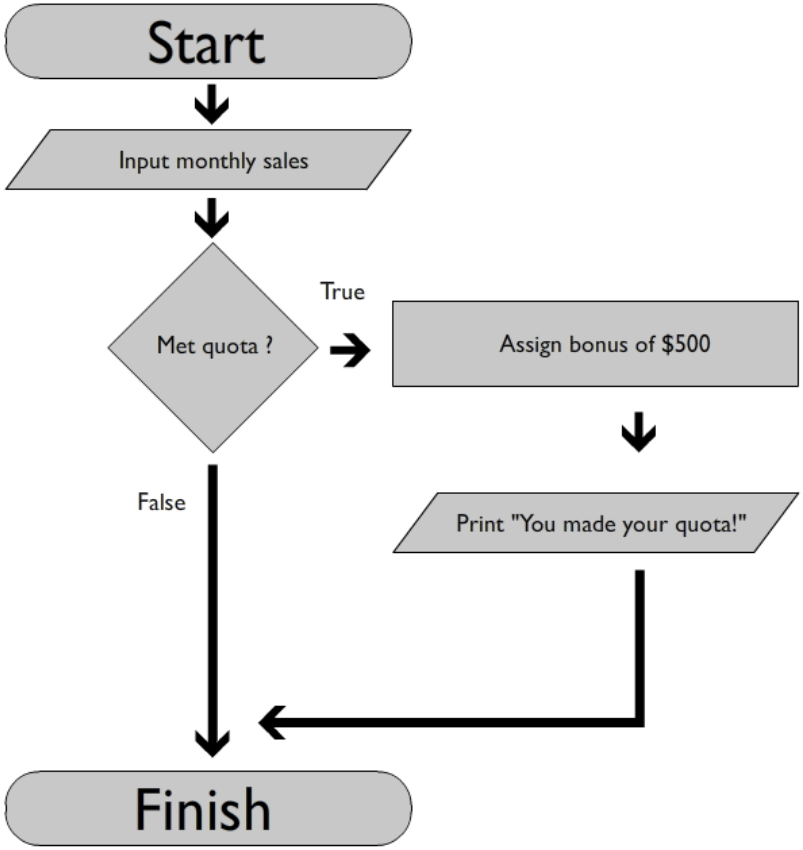
Programming Challenge: Number Guessing Game (part 1)

- Ask the user to guess a number between 1 and 10. Assume they will enter an Integer.
- Pick a number between 1 and 10 that is your "secret" number (for example, 5)
- If the user types in your secret number, tell them that they win!
- If the user types in a number less than or greater than your secret number, tell them that they're either above or below the number and to try again

Programming Challenge: Calculating a bonus

- You're the manager of a large, distributed sales force
- You want to create an easy to use tool that will allow your sales staff to do the following:
  - Input their monthly sales amount
  - Determine if they made their monthly quota of $10,000
  - If they made their quota, they are eligible for a bonus of $500
  - If they made their quota, they should receive a "Good Job!" message
  - At the end of the program you should print out how much their bonus will be ($0 or $500)

Programming Challenge: Calculating a bonus



Extension

- All sales people should receive 1% commission on their sales
- If a sales person made over 50,000, they should receive 5% commission on their sales (instead of 1%) – this is in addition to their $500 bonus for making their quota
- Print out their total take-home amount (bonus + commission) at the end of the program

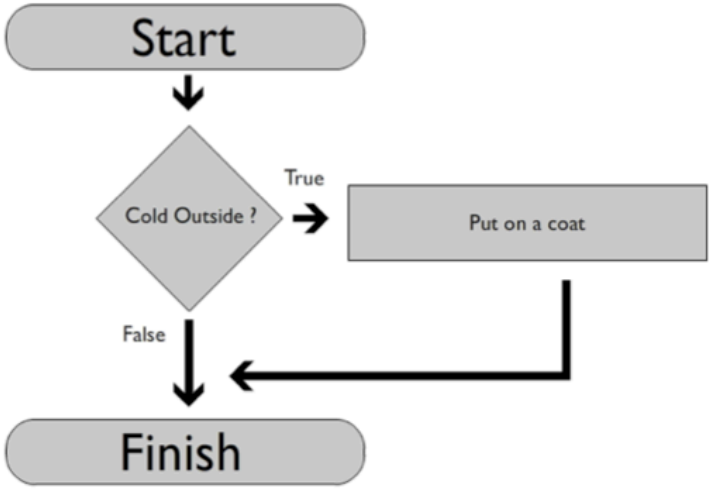
Selection Statements in the Wild!

- How are selection statements used in ATM machines?
- How many selection statements can you count from your last ATM transaction?

The IF – ELSE structure

Simple Selection Statements

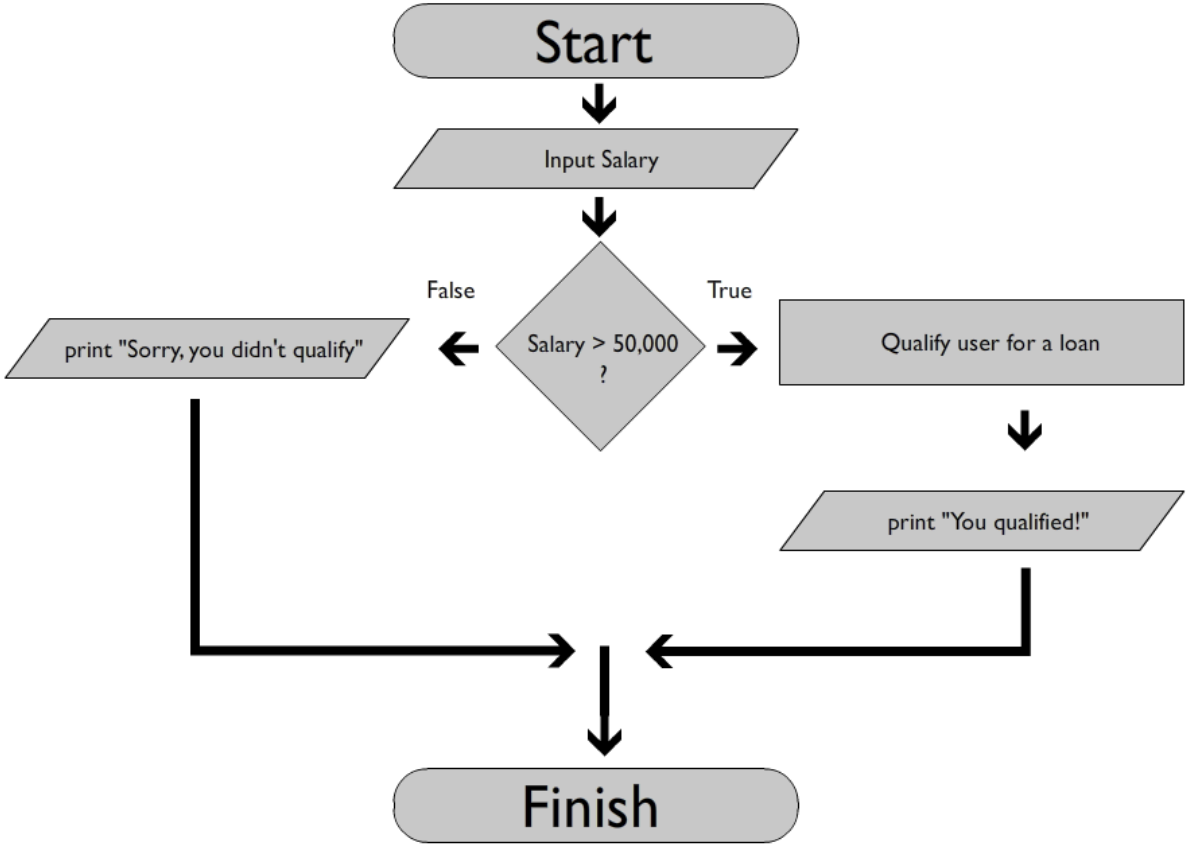
- The selection statements we have been writing so far have only allowed us to create a single alternate branch of execution
- There are many times when we need to create multiple branches of execution based on the value of a Boolean expression



The IF-ELSE structure

- The IF-ELSE structure allows you to perform one set of statements if a condition is true, and another if it is false

The IF-ELSE structure



The IF-ELSE structure

```
if temperature < 32:
  print (“it’s freezing outside!”)

else:
  print (“it’s not so bad outside …”)
```

Programming Challenge: Calculating Overtime Pay

- If a worker works more than 40 hours in a week he or she is entitled to overtime pay.
- Overtime pay is calculated at the rate of 1.5 times the worker's hourly rate.
- This additional rate is only applied to hours worked above the 40 hour limit.

Programming Challenge: Calculating Overtime Pay

- Input: Hourly rate of pay
- Input: Number of hours worked in 1 week
- Process: If the hours worked is less than 40, simply multiply hourly rate by hours worked
- Process: If the hours worked is greater than 40:
  - Multiply hourly rate by hours worked for 40 hours.
  - Subtract 40 from the the total hours to obtain the overtime hours
  - Multiply overtime hours by 1.5 times the rate of pay
  - Add overtime pay to base pay
- Output: Total Pay

+

String Comparison

String Comparison

- So far we have been writing Boolean expressions that evaluate based on numeric data
  - Example: x > 5; y < 10; z == 100
- We can also construct Boolean expressions that can test relationships between strings
- When we compare strings we are essentially reducing them to their zeros and ones and comparing them numerically

Standard ASCII Table

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Boolean Operators for Strings

```
'dog' > 'cat'                  # is 'dog' greater than 'cat' ?

'fish' < 'alligator'           # is 'fish' less than 'alligator' ?

'elephant' == 'tiger'          # are 'elephant' and 'tiger'
                               # equivalent?
'bat' != 'honey badger'
                               # are these strings different ?
'bat' > 'back'
                               # is 'bat' greater than 'back'
```

Programming Challenge: Password Protection

- Write a program that asks the user for a password
- Check to see if the password that was submitted is equal to the string ‘secret’
- If it is, print out a “welcome” message
- Otherwise, tell them to try again

Basic string manipulation

- Python has a huge string manipulation library that allows you to interact with and modify strings. We are going to get more in depth with this package later in the semester.
- For now we will only be exploring two small functions in this package – lower() and upper()
- The lower() function converts the characters in a string to all lowercase, while the upper() function converts the characters in a string to all uppercase
- These functions are not built into the Python library directly, but exist inside the "str" module – as such they must be referred to using "dot syntax"
- Example:
  - `string_lc = str.lower('Harry Potter')  # string_lc = 'harry potter'`
  - `string_uc = str.upper('Harry Potter')  # string_uc = 'HARRY POTTER'`

Programming Challenge: Case insensitive password

- Rewrite your password protection program to be case insensitive (i.e. the password "Secret" will also let you into your program)

Programming Challenge: Alphabetize two strings

- Ask the user to type in two names
- Compare the names and print them out in alphabetical order

String Length

- You can ask Python to count the number of characters contained in a string using the len() function
- len() returns an integer that represents the total length of a string
- Example:

```
myname = 'harry'
print (len(myname))   # 5
```

Programming Challenge: Comparing the size of two strings

- Ask the user to input two names
- Sort the names in size order and print them out to the user

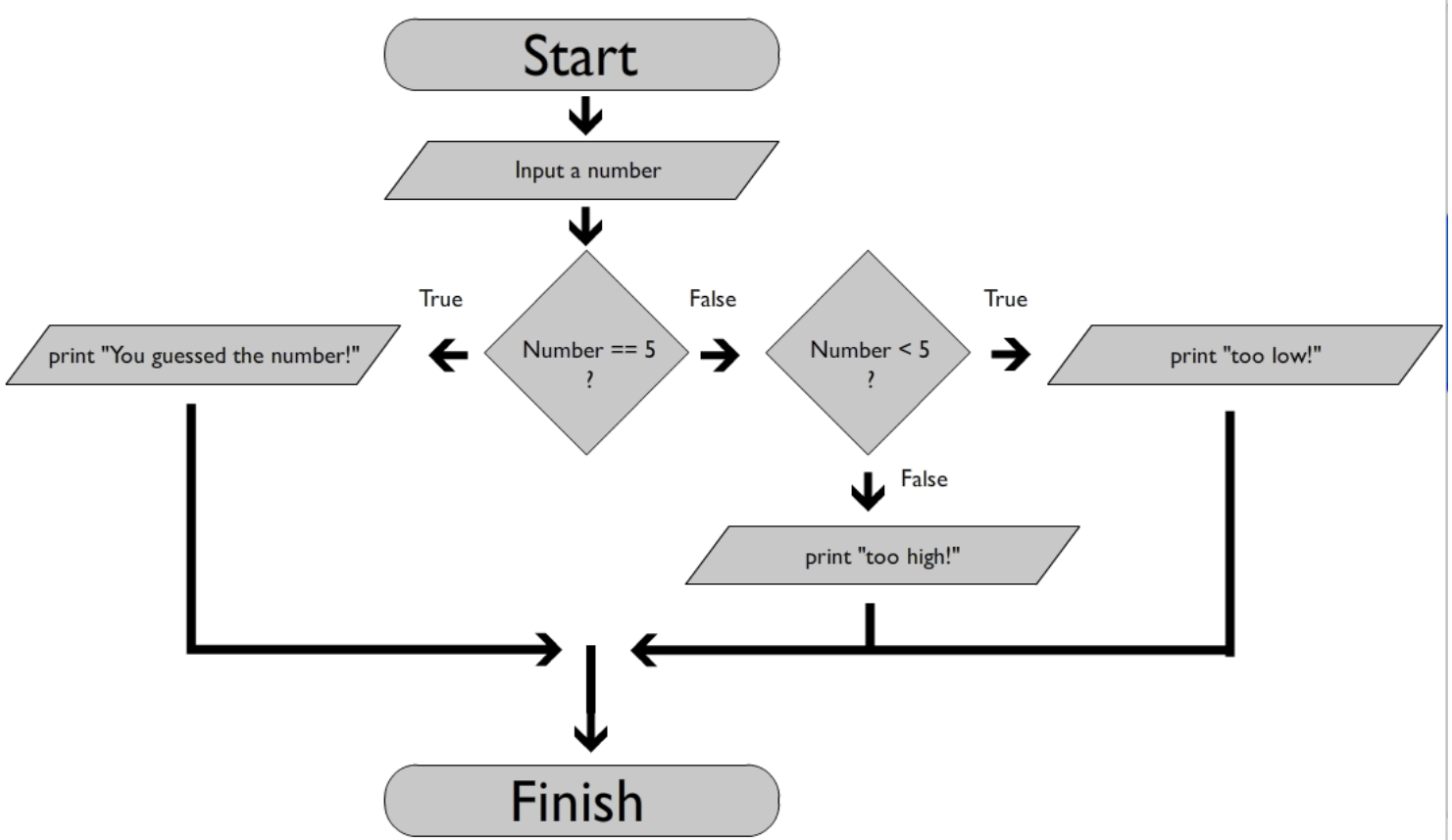
+

Nested Decision Structures

Nested Decision Structures

- Sometimes you need to ask "follow up" questions after you've evaluated the value of a Boolean expression
- Python allows you to "nest" decision structures inside one another, allowing you to evaluate additional conditions

Guess the Number using Nested Decision Structures



Programming Challenge

- Re-write the "guess the number" game using a nested decision structure.
- If the user guesses the number they win. If they don't you should tell them to guess higher or lower next time depending on their answer.

Guess the Number using Nested Decision Structures

```
secretnumber = 5

usernumber = int(input('Guess a number '))

if usernumber == secretnumber:
    print ("you guessed it!")
else:
    if usernumber < secretnumber:
        print ("your number is too low")
    else:
        print ("your number is too high")
```

Nested Decision Structures

- Indentation is key – Python will use the indentation level of a structure to determine its relationship to any previous statements

Programming Challenge: Freezing / Boiling / OK Guppies

- Guppies are hardy fish, but they can't live in all water temperatures.
- The acceptable range for guppies is between 72 and 86 degrees Fahrenheit.
- Write a program that asks the user for a temperature. Then display one of three messages based on the information provided:
  - You're going to freeze your guppy!
  - You're going to boil your guppy!
  - Your guppy is going to be fine!

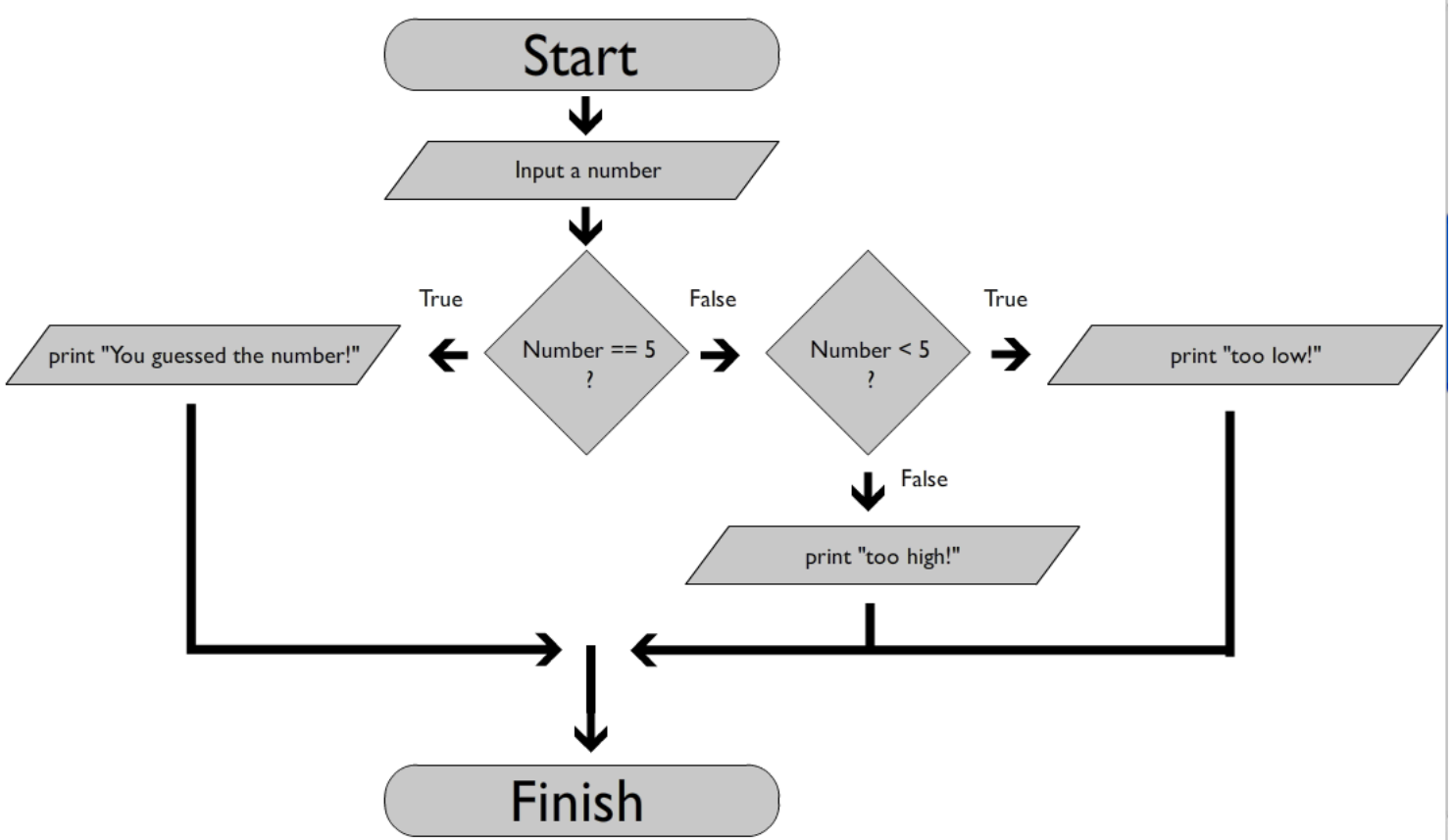
Programming Challenge

- Write a program that asks the user to enter in a number greater than or equal to zero and less than or equal to 100. If they do not you should alert them and end the program.
- Next, determine the letter grade associated with the number. For example, and A is any grade between 90 and 100. Report the letter grade to the user.

Programming Challenge: Loan Qualification

- You're working for a small bank that wants to write a program to allow its customers to pre-qualify themselves for a personal loan
- Rules for qualification are as follows:
  - Borrower must make more than $50,000 per year and be at his or her job for at least 2 years
  - The 2 year job requirement can be waived, however, for borrowers making more than $100,000 per year
- Write a program to ask the user for their yearly salary as well as the # of years they have been at their current company. Use the rules above to output the string 'You qualify' or 'You do not qualify'

Guess the Number using Nested Decision Structures



Guess the Number using Nested Decision Structures

```
secretnumber = 5

usernumber = int(input('Guess a number '))

if usernumber == secretnumber:
    print ("you guessed it!")
else:
    if usernumber < secretnumber:
        print ("your number is too low")
    else:
        print ("your number is too high")
```

Nested Decision Structures

- Indentation is key – Python will use the indentation level of a structure to determine its relationship to any previous statements

+

IF-ELIF-ELSE Structure

Testing a series of conditions

- Testing a series of conditions using an IF-ELSE structure can result in a large amount of indentations
- Sometimes this can cause your code to become difficult to read
- Example: Grade determination program
  - Input: ask the user for a numeric grade (i.e. 95)
  - Process: convert the grade to its letter format (A through F)
  - Output: print the letter grade

Grade Determination Program

```
g = float(input('grade '))

if (g > 90):
    print ('A')
else:
    if (g > 80):
        print ('B')
    else:
        if (g > 70):
            print ('C')
        else:
            if (g > 60):
                print ('D')
            else:
                print ('F')
```

IF-ELIF-ELSE

- You can simplify complex IF statements by using the ELIF structure
- ELIF is an optional structure that can be placed between your IF and ELSE statements
- It allows you to evaluate additional conditions at the same level as the original IF statement

IF-ELIF-ELSE

```
g = float(input('grade '))

if g > 90:
    print ('A')
elif g > 80:
    print ('B')
elif g > 70:
    print ('C')
elif g > 60:
    print ('D')
else:
    print ('F')
```

IF-ELIF-ELSE

- Some notes about using ELIFs:
  - Conditions are tested in the order in which they are written. Once a condition evaluates to True all future conditions are skipped
  - An ELSE statement at the end of a decision structure is considered the "catch all" statement – if all conditions above end up failing then the statements inside the ELSE block will execute
  - However, using an ELSE statement at the end of your decision structure is optional.
  - There is no logical need for an IF-ELIF-ELSE statement. You can always write a program without it by using a standard IF-ELSE block. The advantage of an IF-ELIF-ELSE statement is that your code may end up being be more readable / understandable.

+

Logical Operators

Logical Operators

- All programming languages provide a set of "logical operators"
- These operators can be used to create complex Boolean expressions that evaluate more than one condition at the same time

Logical Operators

```
x = 10
y = 5
a = 20
b = 25

if x > y and a < b:
    print ('yes!')
else:
    print ('no!')
```

Logical Operators

- Logical operators are used to combine Boolean expressions into a composite Boolean expression
- There are three main logical operators that we use regularly in programming
  - and
  - or
  - not

The "and" operator

- "and" can be used to combine two Boolean expressions
- The resulting Boolean expression will evaluate to be True if the two Boolean expressions it is connecting both evaluate to be True

```
True  and True  => True

True  and False => False

False and True  => False

False and False => False
```

Let's evaluate!

```
a = 5

b = 10

print (a > b and a > 1)

print (a > 1 and b > a)

print (a == 5 and b < 100)

print (a > 1 and b < 1 and b > a)

print (a > 1 and b > 1 and b > a)
```

"and" Example

Loan Qualifier

```
salary = float(input('How much do you make? '))
years  = float(input('How long have you been at your job? '))

if salary >= 50000 and years >= 2:
    print ('You qualify for a loan!')

else:
    print ('You do not qualify for a loan')
```

The "or" operator

- "or" can also be used to combine two Boolean expressions
- The resulting Boolean expression will evaluate to be True if EITHER of Boolean expressions it is connecting evaluates to be True

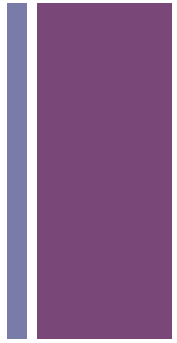
```
True  or True  => True

True  or False => True

False or True  => True

False or False => False
```

Let's evaluate!



```
a = 5

b = 10

print (a > b or a > 1)

print (a > 1 or b > a)

print (a == 5 or b < 100)

print (a > 1 or b < 1 or b > a)

print (a > 1 or b > 1 or b > a)
```

“or” Example

Guppy Temperature

```
temp = float(input('What is the temperature of your fish tank? '))

if temp < 72 or temp > 86:
    print ("The temperature is too extreme!")
```

The “not” operator

- The “not” operator is a unary operator that reverses the logical value of its argument
- This means that it will “flip” a True value into a False value, and vice versa

“not” example

```
username = input('username? ')


if not (username == 'Harry'):
    print("invalid input!")
else:
    print("Welcome, Harry!")
```

Programming Challenge: Username and Password

- Write a program that asks a user for a username and a password
- Check to see if BOTH the username and password are correct
- If so, provide a Welcome message to the user
- If not, provide a Login Failure message to the user