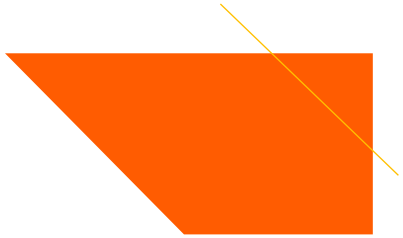
Introduction to Problem Solving in Python

COSI 10A



Class objectives

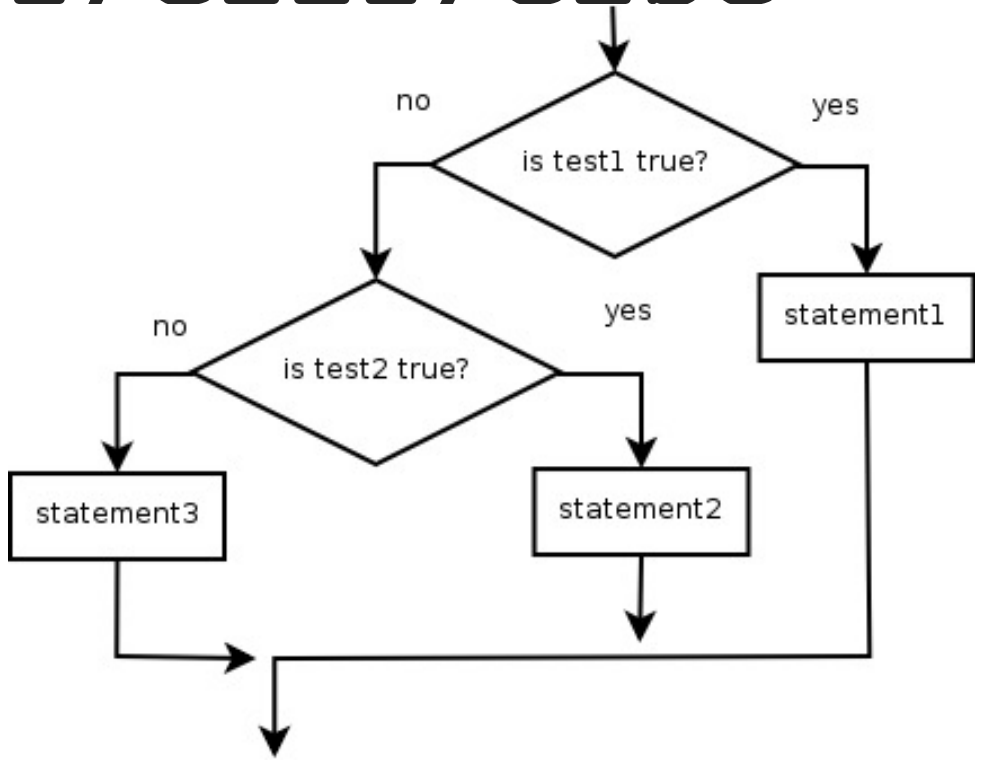
- Boolean Logic
- Strings

Review: Nested `if/elif/else`

- Chooses between outcomes using many tests

Syntax:

```
if test:
    statement(s)
elif test:
    statement(s)
else:
    statement(s)
```



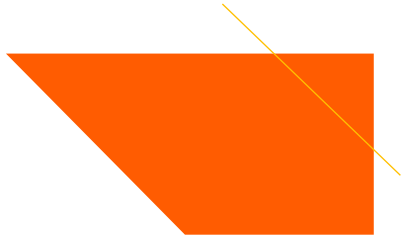
Example:

```
if x > 0:
    print("Positive")
elif x < 0:
    print("Negative")
else:
    print("Zero")
```

Review Nested `if` structures

Exactly 1 path	0 or 1 path	0, 1, or many paths

`if test:`
`  statement(s)`
`elif test:`
`  statement(s)`
`else:`
`  statement(s)`
	`if test:`
	`  statement(s)`
	`elif test:`
	`  statement(s)`
	`elif test:`
	`  statement(s)`
		`if test:`
		`  statement(s)`
		`if test:`
		`  statement(s)`
		`if test:`
		`  statement(s)`



Boolean logic


BOOLEAN HAIR LOGIC
A
B
AND
OR

Type `bool`

- `bool` is a logical type whose values are **True** and **False**
- A logical test is a Boolean expression
- Like other types, it is legal to:
  - create a bool variable
  - pass a bool value as a parameter
  - return a bool value from function
  - call a function that returns a bool and use it as a test

Relational expression

- `if` statement use logical tests

```
if i <= 10: …
```

- Tests use **relational operators**

Operator	Meaning	Example	Value
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

Logical operators

- Tests can be combined using **logical operators**

Operator	Description	Example	Result
`and`	and	`(2 == 3) and (-1 < 5)`	`False`
`or`	or	`(2 == 3) or (-1 < 5)`	`True`
`not`	not	`not (2 == 3)`	`True`

- "Truth tables"

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

p	not p
True	False
False	True

Evaluating logical expressions

- Relational operators have lower precedence than math; logical operators have lower precedence than relational operators

```
5 * 7 >= 3 + 5 * (7 - 1) and 7 <= 11
5 * 7 >= 3 + 5 * 6 and 7 <= 11
35    >= 3 + 30 and 7 <= 11
35    >= 33 and 7 <= 11
True and True
True
```

Logical questions

- What is the result of each of the following expressions?

```
x = 42
y = 17
z = 25

y < x and y <= z
x % 2 == y % 2 or x % 2 == z % 2
x <= y + z and x >= y + z
not(x < y and x < z)
(x + y) % 2 == 0 or not((z - y) % 2 == 0)
```

Type bool

```
minor     = age < 21
is_prof   = "Prof" in name
loves_cs = True

# allow only CS-loving students under 21
if minor and not is_prof and loves_cs:
          print("You can have ice-cream!")
```

Testing multiple conditions

- You often will find yourself wanting to test more than one condition

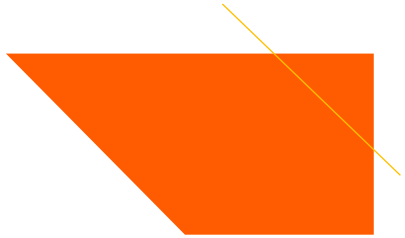
Example:

```
if age1 >= 18:
  if age2 < 18:
    do_something()
```

```
if age1 >=18 and age2 < 18:
    do_something():
```

- `and` operator forms a test that requires that both parts of the test evaluate to `True`
- If you want to test whether a variable is equal to 1 or 2 you use the `or` operator

```
if num == 1 or num == 2:
        process_number(num)
```



Strings

Strings

- A **string** is a type that stores a sequence of characters

Syntax:

```
name = "text"
name = expression
```

Example:

```
name = "Daffy Duck"
```

Example:

```
x = 3
y = 5
point = "(" + str(x) + ", " + str(y) + ")"
```

Strings

- A **string** is an object of type `str` (instance of the `str` class)

- **+** operator when used on strings produces the concatenation of those strings
- ***** operator when used on a string and an integer produces repeated copies of the string

```
s1 = "hello"
s2 = "there"
combined = s1 + " " + s2  #'hello there'
repeated = s1 * 3         #'hellohellohello'
```

How long is my string?

Syntax:

```
length = len(string)
```

Example:

```
s = "Hi there!"
count = len(s)     # 9
```

Index

- Each character in a string is associated with a unique integer called **index**

```
s1 = "Hello"
```

index	0	1	2	3	4
character	H	e	l	l	o

```
s2 = "how are you?"
```

index	0	1	2	3	4	5	6	7	8	9	10	11
character	h	o	w		a	r	e		y	o	u	?

- First character's index : 0
- Last character's index : 1 less than the string's length

Index

- How can you access the last character in a string `s`? `len(s) - 1`

- As an alternative, Python uses negative numbers to give easy access to the chars at the end of the string

```
s = "how are you?"
```

index	0	1	2	3	4	5	6	7	8	9	10	11
character	h	o	w		a	r	e		y	o	u	?
	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- Last character's index : -1

Accessing characters in a string

- To access a specific character of a string use square bracket

```
s1 = "Hello"
s1[0] # 'H'
S1[3] # 'l'
```

index	0	1	2	3	4
character	H	e	l	l	o

Accessing substrings

Syntax:

```
part = string[start:stop]
```

Example:

```
s = "Merlin"
mid = s[1:3]   #er
mid = s[:3]    #Mer
mid = s[1:]    #erlin
```

String functions

Method name	Description
`find(`**`str`**`)`	index where the start of the given string appears in this string (-1 if not found)
`lower()`	a new string with all lowercase letters
`upper()`	a new string with all uppercase letters

- These methods are called using the dot notation below:

```
starz = "Biles & Manuel"
print(starz.lower())    # biles & manuel
```

Modifying strings

- String operations and functions like `lowercase` build and return a new string, rather than modifying the current string

```
s = "Test"
s.upper()
print(s)    # Test
```

Strings are immutable. The value cannot change

- To modify a variable's value, you must reassign it:

```
s = "Test"
s = s.upper()
print(s)    # TEST
```