enactus
University of Sheffield

CODE CREATORS
SHEFFIELD

2024 PYTHON BEGINNER COURSE

Lesson 2

SHALEV LAU
/in/shalevl
/mushroomhater07

Lesson 2 - Content

enactus University of Sheffield

CODE CREATORS SHEFFIELD

1.What is Control Flow?
2.IF Statements
3.Assignment vs. Equality vs. Identity
4.Lecture 2 Mini-Project: Odd or Even
5.Other comparison operators
6.Boolean Operators
7.Boolean Algebra
8.Short Circuiting
9.Lecture 2 Mini-Project 2:FizzBuzz!
10.Ternary Operators (In-Line If)
11.Lecture 2 Mini-Project 3: Days of The Week
12.Structural Pattern Matching (Switch Statements)
13.Lecture 2 Project: Love Calculator 🥰

What is Control Flow ?

Is the concern of the order in which instructions are executed. Code is ran from the first to last line unless the computer comes across a control flow statement. Today we will be looking at branching: conditional structures (**if statements**) and **switch statements**. In the coming lectures we will look at loops, and functions. *Here's the founder of Microsoft to explain an if statement:*

enactus
University of Sheffield
CODE CREATORS
SHEFFIELD

Branching

To change the order of how instructions are executed we need to branch. Branching is an instruction causes a computer to begin executing a different instruction sequence and thus deviate from its default behaviour of executing instructions in order.

In low-level languages, this is a command you have to specify with the memory address of the instruction (see below). In any high-level language any control flow statement and function calls will implicitly cause branching.

LMC ASSEMBLY

```
INP
STA A
INP
STA B
LDA A
SUB B
BRP isPositive # Branch if the
value in the register > 0.
LDA B
OUT
HLT
isPositive LDA A
HLT
A    DAT
B    DAT
```

enactus University of Sheffield

CODE CREATORS SHEFFIELD

IF Statements

Check the condition provided for truthiness and executes the corresponding branch if true. Optionally, you can check any number of extra conditions specified in elif branch. Finally, you can specify a default if none of the other conditions were met with an else branch.

```
if False:  # You 'd place a condition here instead.
  pass
elif False: # There's no point hard coding truthy or falsy values.
  pass
else True: # In this case the code in the else branch will always
execute, which misses the point of branching.
  pass
```

Every piece of code belonging to the branch must be indented (usually 2 spaces or a tab). Branches must always have code otherwise you will see a IndentationError: expected an indented block, or remove the branch completely. If you don't want to, you can use a null operation (*pass*). It's used as a placeholder for future implementation of functions, loops, etc which requirement a statement syntactically.

enactus University of Sheffield

CODE CREATORS SHEFFIELD

What Makes up a Condition?

A boolean (or equality) expression is evaluated for a computer to decide to true or false. It takes into account the truthiness of values. Remember in Lecture 0 when we talked about truthy and falsy values?

Let's see how we can make a condition.

Assignment vs. Equality vs. Identity

As you've learnt in the last lecture, a single `=` last lecture performs variable assignment.

If we want to check if 2 values are equal to each other (check for equality) we need `==`. Expressions are first evaluated before the check.

```
x = 5 # Variable assignment. We're giving x the value of 5.
if x == 5:
  print(f"{x} if obviously equal to 5.")
  print(x == 5 == 2 + 3) # True
```

```
print("5" == 5 == 5) # is false.
```

Even though they're they represent the same thing, they have different data types. Notice how if only one of the equalities is false, the rest of equality is false because Python's stops evaluation. We will look at this behaviour, known as short-circuiting in part 11.

Oppositely, you can check for inequality to check if they aren't of equal values.

```
print("pham" != "mid" ) # True
```

enactus University of Sheffield

CODE CREATORS SHEFFIELD

Identities: The IS Operator

In the advanced course, you'll understand how Python values are referenced from the heap and hence the important of is

For now understand that even though 2 variables have the same value, they don't reference the same object in memory. Recall that every value in Python is an object!

Remember in Lecture 1 when we talked about operands?
You can think of the identity operator as checking that.

Lecture 2 MINI-Project 1: ODD OR EVEN

An even number is defined as divisible by 2 with no remainders. Odd numbers are just numbers that aren't even.

Given that you know the modulus operator (%) gives the remainder, make a program to determine whether any real number is odd or even.

Worked solutions will be posted in *5 minutes*.

enactus University of Sheffield

CODE CREATORS SHEFFIELD

Solution

Did you manage to do it? If not, don't stress. We'll walk you through it!

```
print("Find out if your real number's odd or
even.\n")
num = int(input("Enter a number: "))

if (num % 2 == 0):
  print("%s is even." % num)
elif (num % 2 != 0):
  print("{} is odd.".format(num))
```

enactus University of Sheffield

CODE CREATORS SHEFFIELD

Optimization

Optimization is the process of modifying a system to make some features of it work more efficiently or use fewer resources.

We can save on one evaluation by simply replacing the `elif` with an `else` branch. This makes a negligible difference to performance, however you can agree it's more readable.

Why can we do this? Numbers can only be odd or even so we only need to check for one condition or the other. negligible

```
print("Find out if your 's odd or even.\n")
num = int(input("Enter a number: "))

if (num % 2 == 0):
  print("%g is even." % num)
else:
  print("{:g} is odd.".format(num))
```

enactus University of Sheffield

CODE CREATORS SHEFFIELD

Light Alternate

You can call `input()` directly in the condition and you can get rid of the intermediary `num` variable. Further you can use f-strings to make the printing more concise.

The take home point is that there are no definite solutions in programming. It's up to you, the developer, to come up with the most optimal with the compromise of readability.

In fact some argue that the below is less readable because we don't know what the input is concerning.

Readability almost always triumphs code conciseness. We have plentiful storage, so saving a few bytes on characters in the source code is NOT worth it. (Not saying your code should be verbose either.)

```
print("Find out if your real number's odd or even.\n")

if (float(input("Enter a number: ")) % 2 == 0):
  print(f"{num:g} is even.")
else:
  print(f"{num:g} is odd.")
```

enactus University of Sheffield

CODE CREATORS SHEFFIELD

Other Comparison Operators

We've looked at equality, inequality, identity comparisons but there are a few more be aware of. The below behave like they do in maths, so don't stress.

Operator	Description
>	Less than
<	Greater than
>=	Less than or equal to
<=	Greater than or equal to

Boolean Operators

You can combine multiple checks in one condition using one of the below. You can model the examples in a truth table too.

Operator	Description	Example
and	Conjunction of values results in true if all values are true.	• `True and True == True` • `True and False == False` • `False and False == False`
or	Disjunction of values results in true given at least one of the values is true.	• `True or False == True` • `False or False == False` • `True or True == True`
xor (not actually a Python boolean operator but you can construct it using its definition)	Exclusive OR results in true if EXACTLY one of the values is true.	• `(A and not B) or (not A and B)`
not	Negation inverts boolean expressions from true to false and vice versa.	• `!True == False` • `!False == True`

enactus University of Sheffield

CODE CREATORS SHEFFIELD

Boolean Algebra

This branch of mathematics deals with operations on logical values.
What we're concerning about is how we can use these rules to write conditions in their simplest forms to improve readability.
We will cover:

- Law of Commutation
- Double Negation
- Association

The rest of the topic is beyond the scope of our course, but I'd like to make you aware of:

- Absorption
- Complements
- De Morgan's Law
- Distribution
- Karnaugh Maps

In the following slides assume A is one condition and B is another.
We will provide analogies to help understand it.

Why is this topic important ?

Amateurish code is very apparent to any experienced dev.
We want to make sure you don't write silly lines like:

```
if raining and cold or cold and raining:
  print("Wear a jacket.")
# Applying the Law of Distribution:
if raining and cold:
  print("Wear a jacket.")
```

Or something you're might not be aware of...

```
if is_sunny == True:
  print("Put on sunglasses. 😎")
# If statements are already
# checks for truthiness,
# so you don't need
# to compare it to True.
if is_sunny : print("Put on sunglasses. 😎")
```

enactus University of Sheffield

CODE CREATORS SHEFFIELD

Law of Commutation

The order of application of 2 separate terms is irrelevant given the same operation is applied to them.

```
A = True
B = False
print(A or B == B or A) # True
print(A and B == B and A) # True
```

It's the same law of commutation as in maths, so the below isn't valid.

```
A = False
B = True
print(A and B == B or A) # False
```

This is like saying 0 x 1 = 1 + 0, which is false.
If the same operation isn't applied to them, obviously the evaluated value will be different.
Real-life analogy: "Shalev and Nedjm are teaching Python." is the same as "Nedjm and Shalev are teaching Python"

Double Negation

Just like in maths where a negative x a negative is a positive, negating a negated value returns its original value.

```
A = True
B = False
print(not A) # False
print(not not A) # True, i.e. A's original value.
print(not B) # True
print(not not B) # False, i.e. B's original value.
```

Real-life analogy: "I don't *not* love Python" is the same as "I love Python".

enactus
University of Sheffield

CODE CREATORS
SHEFFIELD

Association

The same operation being performed on variables will results in the same value irrespective of the grouping of variables.

```
A = True
B = False
print(A or B or (A or B) == (A or B) or A or B) # True
```

Like if you do 1 + (2 + 3), it doesn't matter if you do the 2 + 3 first.

```
A = False
B = True
print( (B and (B and A) and B) == ( (B and B) and A and B) ) # True
```

You must place an extra pair of brackets around each expression because the short-circuiting nature of boolean expression evaluation contradicts the documented operator order precedence. This is an annoying bug we found in Python while we were researching!

Real-life analogy: "Nedjm and his homies, Shalev and Haris are coming to the crib" is the same as "Shalev and Haris and their homie Nedjm are coming to the crib" and also the same as "Nedjm, Shalev and Haris are coming to the crib"

enactus
University of Sheffield

CODE CREATORS
SHEFFIELD

Short Circuiting

This doesn't mean Python causes your computer to malfunction. 💀

We mean the stoppage of execution of Boolean operation if the truth value of expression has been determined already, because the rest of the evaluation is redundant. The evaluation of expression takes place from left to right.

To prove this behaviour exists run this code:

```
def return_five():
  print("5 was returned.")
  return 5

print([] and return_five())
```

If the whole expression, were evaluated you should've seen "5 was returned" output to the screen. However, because you don't it implies the function was never called and therefore never evaluated?

Why? An empty list is falsey (as discussed last lecture), therefore it evaluates to `False` and because `and` requires both operands to be `True` to evaluate to `True`, it can just default to `False` instead since Python already knows it can never be `True` due to the falsey value.

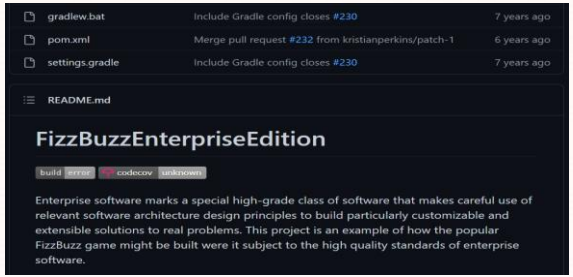
Can you guess what will happens if you switch it to an `or` operator?

enactus University of Sheffield

CODE CREATORS SHEFFIELD

Lecture 2 MINI-Project 2: FIZZBUZZ!

This has become a meme in the software engineering world because it's the simplest question an interviewer can ask you. In fact, someone went to the labour of making it into enterprise ready software for the lols. 😂 CTRL + CLICK on the image to see the repo.



For numbers 1 to 100, for every multiple of 3 print "Fizz" and for multiples of 5, print "Buzz" and for multiples of both 3 and 5, print "FizzBuzz!" otherwise just print the value of `num`. We've already started the code for you, all you need to do is fill in the control flow logic. Worked solutions will be posted in *5 minutes*.

```
for num in range(1, 101): # We will go over loops next lecture.
  # Your code goes here.
  # Remember that you need to indent it so it belongs
  # to this for loop.
```

enactus University of Sheffield

CODE CREATORS SHEFFIELD

A Solution

Remember to code along, because it gets you to think.

```
for num in range(1, 101):
    if num % 3 == 0 and num % 5 == 0:
        print("FizzBuzz!")
    elif num % 3 == 0:
        print("Fizz")
    elif num % 5 == 0:
        print("Buzz")
    else:
        print(num)
```

Maths brainiacs would've realized you can simplify `num % 3 == 0 and num % 5 == 0` into `num % 15 == 0`.

Why? 15 is the (lowest common) multiple of 3 and 5, and therefore if the number's divisible by 15, it's also divisible by 3 and 5 so we have a FizzBuzz!

It's important you check for "FizzBuzz!" first otherwise the program will either only print "Fizz" or "Buzz" (or the number), depending on the branch you write first.

Important: Once a condition is satisfied for a branch, the rest of them are ignored.

enactus University of Sheffield

CODE CREATORS SHEFFIELD

ONE-LINE Solution

Our homie Shalev always loved writing his code all on one line to troll our CS teacher. Here's a tribute to him:

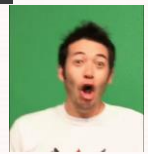
```
for num in range(1,101): print("FizzBuzz" if num % 15 == 0 else "Fizz" if num % 3 == 0 else "Buzz" if num % 5 == 0 else num)
```

I know your mind's blown right now, but this a good opportunity to introduce you to ternary operators. 🤯 BTW, don't write code like this, your colleagues will hate you.

While you can write a single line of code belonging to a block all on one line, let's kick it onto the next line so it's slightly easier to read.

```
for num in range(1,101):
  print("FizzBuzz" if num % 15 == 0 else "Fizz" if num % 3 == 0 else "Buzz" if num % 5 == 0 else num)
```

Let's analyse `"FizzBuzz" if num % 15 == 0` closer, but first we need to know about ternary operators.



enactus University of Sheffield

CODE CREATORS SHEFFIELD

Ternary Operator(IN-LINE IF)

Are conditional expressions which behave like if statements but evaluate to some value (here it's "nice.") based on some value being true (i.e. she is_nice) otherwise it must evaluate to some other value (she is "not nice." 😡).

```
is_nice = True
description = "nice." if is_nice else "not nice.")
# value_if_true if condition else value_if_false
print("Alexandra is " + description)
```

You **cannot** omit the alternate condition (i.e. *else* "not nice.") otherwise you get a SyntaxError: invalid syntax. The name "ternary" means "composed of 3 parts", so you can not miss one part of the operator – they're all important.

THIS IS NOT A REPLACEMENT FOR IF STATEMENTS, JUST A SHORT AND DIRTY ALTERNATIVE.

Before Python 2.5, this wasn't a native feature. You may (rarely) see them written as a tuple or list accessed by using a boolean index (which evaluates to 0 for False and 1 for True). You'll understand lists, tuples and why this works in Lecture 3.

It's **not** recommended you write ternary operators this way because the code's purpose isn't very clear so it isn't "Pythonic" – conforming to the PEP8 style guide for writing clean code.

```
is_nice = True
print("Alexandra is " + ("not nice.", "nice.")[is_nice]) # (value_if_false,
value_if_true)[condition]
```

enactus University of Sheffield

CODE CREATORS SHEFFIELD

FIZZBUZZ Ternary Operator Usage

Now that we know about ternary operators, we can explaining the one-line FizzBuzz program. In the case `num` multiple of 3 and 5 (15), the ternary operator expression evaluates to `"FizzBuzz"` and that gets passed into `print()` to be output.
Now when the code is run again, we get a different value of `num` thanks to the for loop. We check if it's divisible by 15, now instead of specifying an alternate hard-coded value we chain on another ternary operator. This is completely valid, since the expressions all evaluate to some value Python can output in the end.
Apart from this wild syntax, the logic for checking whether to output FizzBuzz is the same.

If you still don't get it, let's work through an example:
Suppose `num` is 25.

```
."FizzBuzz" if num % 15 == 0 else # …
```

"`num`, are you divisible by 15? No, okay I'll check the alternate value."
Hey look, the alternate value is another ternary operator. Let's do it again!

```
"Fizz" if num % 3 == 0 else # …
```

"Is 25 divisible by 3? I don't think so, let's check the alternative.
"OMDs, it's yet another ternary operator. Let's do what we just did."

```
"Buzz" if num % 5 == 0 else num
```

"Hey, 25 is divisible by 5, since you said you I'll evaluate to "Buzz" if that's true so now I can chill and stop evaluating the expression otherwise I'd just given you the original number, `num`

enactus
University of Sheffield
CODE CREATORS
SHEFFIELD

Lecture 2 MINI-Project 3: DAY OF THE WEEK

For practice, I want you to accept a day of the week (e.g. Sunday) as input and output to the user whether that day is a weekday or weekend.

Reminder that weekdays are:

- Monday
- Tuesday
- Wednesday
- Thursday
- Friday

Weekends:

- Saturday
- Sunday

Worked solutions in *5 minutes*.

Example output:

```
Enter a day of the week: sunday
Sunday is a weekday.
```

- **Hint 1:** `str.lower()` can be used to put the user input into lower case so your program isn't affected by case-sensitivity.

enactus University of Sheffield

CODE CREATORS SHEFFIELD

A (BAD) Solution

We'll excuse you for writing code like this since you're beginners.

Remember that *Beautiful [code] is better than ugly*, as stated in the Zen of Python. This is not beautiful because it's hard to read and repetitive.

```
day = input("Enter a day of the week: ").lower()

if (day == 'monday'):
  print(f"{day.title()} is a weekday.")
elif (day == 'tuesday'):
  print(f"{day.title()} is a weekday.")
elif (day == 'wednesday'):
  print(f"{day.title()} is a weekday.")
elif (day == 'thursday'):
  print(f"{day.title()} is a weekday.")
elif (day == 'friday'):
  print(f"{day.title()} is a weekday.")
else:
  print(f"{day.title()} is a weekend.")
```

An OK Solution

We'll excuse you for writing code like this since you're beginners.

Remember that *Beautiful [code] is better than ugly*, as stated in the Zen of Python. This is not beautiful because it's hard to read and repetitive.