Python programming and data visualization for beginners

Dr Joel Martin

Week 2

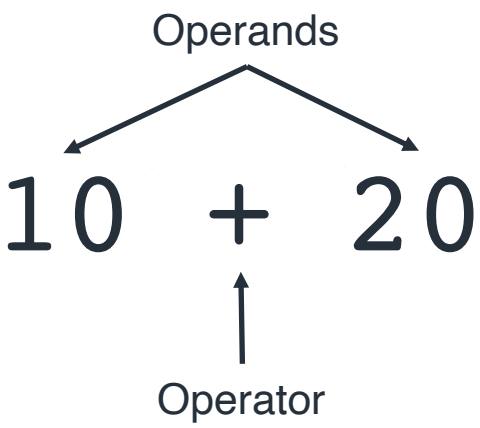
Operators, control flow statements, defining functions

- Operators
  - Logical, arithmetic, assignment, comparison, membership, identity, bitwise
- Control flow statements
  - if, while, for
- Defining functions
  - def

Operators

What are they?

- Special symbols designating some sort of computation to be performed
- Python has the following kinds of operator:
  - Logical (e.g., `and`, `or`, `not`)
  - Arithmetic (e.g., `+`, `-`, `/`, `*`)
  - Assignment (e.g., =)
  - Comparison (e.g., ==, >, <)
  - Membership (`in`, `not in`)
  - Identity (`is`, `is not`)
  - Bitwise (e.g., `&`, `|`, ~)



Operators

Logical

- Used to join expressions in a Boolean context
- `not` 

  - Return False if the result is `True` (i.e., reverse the result)
- `and` 

  - Return `True` if both statements are `True`
- `or` 

  - Return `True` if at least one statement is `True`

```python
# Some Boolean expressions
world_is_round = True
world_is_flat = False
cake_is_yummy = True
is_saturday = True
is_sunday = False

# Boolean statements (all get printed)
if cake_is_yummy and world_is_round:
    print("Cake is yummy and the world is round")

if is_saturday or is_sunday:
    print("Don't go to work")

if not world_is_flat:
    print("The world is not flat")
```

Operators

Arithmetic

- For performing mathematical computations
- Modulo returns the remainder of a division
  - `10 % 6 = 4`
  - `42 % 8 = 2`
- Floor division returns the largest integer less than or equal to the result of a division
  - `10 // 6 = 1`
  - `42 // 8 = 5`

```
# Arithmetic operators in action!
celsius = (fahrenheit - 32) * (5 / 9)
```

Operator	Name	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulo	a % b
**	Exponentiation	a ** b
//	Floor division	a // b

Operators

Assignment

- Used to assign values to variables
- = is the main assignment operator
- The rest are **augmented assignment operators**, which are simply a shorthand for updating a variable value in-place
- For now, stick with whichever approach makes the most sense to you

Operator	Example	Equivalent
`=`	`a = 10`	`a = 10`
`+=`	`a += 6`	`a = a + 6`
`-=`	`a -= 6`	`a = a - 6`
`*=`	`a *= 6`	`a = a * 6`
`/=`	`a /= 6`	`a = a / 6`
`%=`	`a %= 6`	`a = a % 6`
`//=`	`a //= 6`	`a = a // 6`
`**=`	`a **= 6`	`a = a ** 6`
`&=`	`a &= 6`	`a = a & 6`
`|=`	`a |= 6`	`a = a | 6`
`^=`	`a ^= 6`	`a = a ^ 6`
`>>=`	`a >>= 6`	`a = a >> 6`
`<<=`	`a <<= 6`	`a = a << 6`

Operators

Comparison

- Used to compare object equality
- Typically used in a Boolean context to control the flow of of a program
- Outcome of a comparison depends on the nature of the data being compared
- Some types can not be compared (e.g., `int` and `str`)

Operator	Name	Example
`==`	Equal	`a == b`
`!=`	Not equal	`a != b`
`>`	Greater than	`a > b`
`<`	Less than	`a < b`
`>=`	Greater than or equal to	`a >= b`
`<=`	Less than or equal to	`a <= b`

Operators

Membership and identity

- Use `in` and `not in` to check whether a specified value is a constituent member or element of a sequence
- Use `is` and `is not` to check whether two variables have the same identity (i.e., they refer to the same object in memory)
- Note that identity and equality are not the same thing

```python
# Compare membership
list_of_nums = [1, 2, 3, 4, 5]
value = 3
if value in list_of_nums:
    # This will get printed
    print(f"{value} is in list_of_nums")

elif value not in list_of_nums:
    # This will not get printed
    print(f"{value} is not in list_of_nums")


# Compare identity
this_thing = None
if this_thing is None:
    # This will get printed
    print("this_thing has the same identity as None")
    # Proof
    print(id(None)
    print(id(this_thing))
```

Operators

Bitwise

- Used to perform Boolean logic on individual bits
- Good to know about, safe to forget (for now)
- Possible to use them in place of logical operators (`and`, `not`, `or`)
- But this is less readable and may not work as expected in all cases

Operator	Name	Example
`&`	Bitwise AND	`a & b`
`|`	Bitwise OR	`a | b`
`^`	Bitwise XOR (exclusive OR)	`a ^ b`
`~`	Bitwise NOT	`a ~ b`
`<<`	Bitwise left shift	`a << b`
`>>`	Bitwise right shift	`a >> b`

Control flow statements

`if / elif / else`

- Used for conditional execution
- `if` statements can be extended with any number of `elif` ('else if') clauses and optionally ended with an `else` clause
- **Short circuit evaluation** – as soon as a true statement is found, the relevant code is executed, and the program continues

```python
# Some Boolean expressions
today_is_weekday = True
today_is_saturday = False
today_is_sunday = False

# Decide what to do
if today_is_weekday:
    print("Go to work.")
elif today_is_saturday or today_is_sunday:
    print("Don't go to work.")
else:
    print("What planet is this?")
```

Control flow statements

`while`

- Used for repeated execution for as long as a condition remains true
- May include the `break` and `continue` statements (not shown here)
- `break` terminates the loop without executing the `else` clause (if present)
- `continue` skips the rest of the code for the current iteration and goes back to testing the initial expression

```python
# Countdown to rocket launch
counter = 5

while counter > 0:
    print(counter)
    counter -= 1
else:
    print("Blast off!")
```

Control flow statements

`for`

- Used to iterate over the elements of a sequence (or some other *iterable* object) in the order that they appear
- At each iteration the current value from the sequence is assigned to a target variable
- Optional `else` to be executed when the sequence is exhausted
- May also include the `break` and `continue` statements (not shown here)

```python
# Iterate over a list of numbers
for number in [1, 2, 3, 4, 5]:
    print(number)
else:
    print("No more numbers in the list")

# Iterate over the characters of a string
for character in "Hello, World!":
    print(character)
else:
    print("No more characters in the string")
```

Defining functions

`def`

- Functions are self-contained blocks of code that encapsulate a specific task or related group of tasks
- They typically take a list of arguments, perform some sort of operation, and then return a result
- Use the keyword `def` to define a function
- If you find that you are repeating the same bits of code over and over again, define a function!