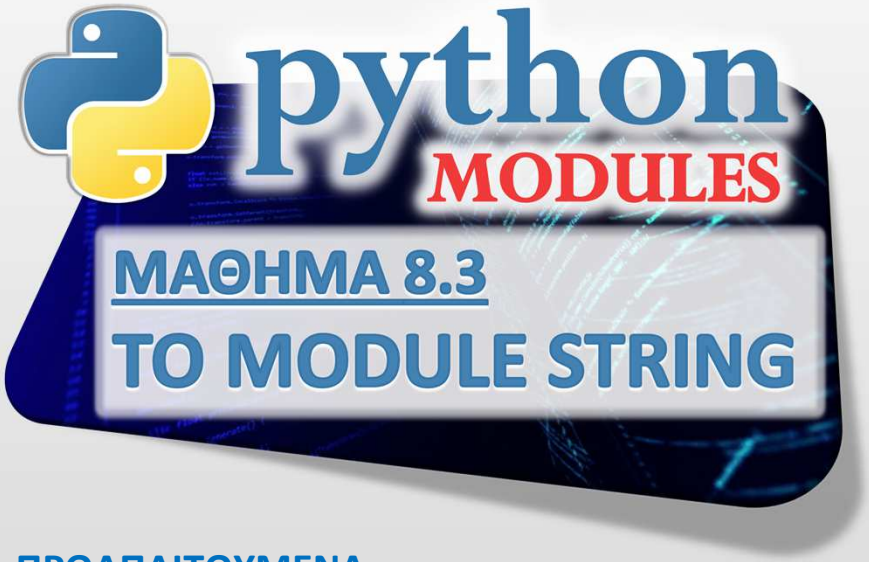

ΠΡΟΑΠΑΙΤΟΥΜΕΝΑ:

1. Python Advanced: Μάθημα 2 - Iterators

ΠΕΡΙΕΧΟΜΕΝΑ:

1. Σταθερές
2. Συντακτικό της format()
   1. Σύνταξη πεδίων αντικατάστασης
   2. Προσδιοριστές Μορφοποίησης
3. Η κλάση Template

Δημήτρης Δ.
Σμαραγδένιος Χορηγός Μαθήματος

Πάνος Γ.
Ασημένιος Χορηγός Μαθήματος

ΜΑΘΗΜΑ 8.3: Το module string

1. Σταθερές

modules psounis

Το **module string**:

- Περιέχει σταθερές που ομαδοποιούν χαρακτήρες και
- Δύο κλάσεις (Formatter και Template) μέσω των οποίων μπορεί να γίνει μορφοποίηση συμβολοσειρών.

Σταθερές:

- Οι ακόλουθες σταθερές ομαδοποιούν χαρακτήρες ως συμβολοσειρές:

Σταθερά	Επεξήγηση
ascii_lowercase	Μικροί ASCII χαρακτήρες
ascii_uppercase	Κεφαλαίοι ASCII χαρακτήρες
ascii_letters	ascii_lowercase + ascii_uppercase
digits	Ψηφία
hexdigits	0-9, A-F, a-f
octdigits	0-7
punctuations	σημεία στίξης
printable	Εκτυπώσιμοι χαρακτήρες (digits, ascii_letters, punctuation, whitespace)
whitespace	περιέχει (μεταξύ άλλων): κενό, tab, linefeed, return, formfeed, vertical tab

Παράδειγμα 1: constants

```
if 'a' in string.printable:
    print(True)
```

Μορφοποίηση συμβολοσειράς (βλ και Python - Μαθ. 9, 2.6)

- Η μορφοποίηση συμβολοσειράς με τη format() προσφέρει πλούσια παραμετροποίηση ώστε να μπορούμε να τυπώνουμε με πλήθος ψηφίων αριθμών, στοίχιση, μετατροπή αριθμών σε άλλα αριθμητικά συστήματα κ.α.
- Είδαμε παραδείγματα με τη format:

Παράδειγμα 2: str.format (από μάθ.9)

```
print("int value: {} {}".format(1, 5+1))
print("int width(right align): {:>10d} ".format(1))
print("int width(left align): {:<10d} ".format(1))
print("float: {}".format(1/3))
print("float: decimals: {:.4f} {:.2f}".format(1/3, 1/8))
print("string: {}".format("Hello there! "))
```

- Στο module string ορίζεται το συντακτικό που ακολουθείται μέσα στη συμβολοσειρά, ορίζοντας μια μικρή γλώσσα αντικαταστάσεων (θα το δούμε αναλυτικά, στο μάθημα αυτό).
- Πρόσθετα η κλάση Formatter έχει τη μέθοδο format(str, *args, **kwargs) που έχει το ίδιο ακριβώς αποτέλεσμα με την str.format(*args, **kwargs)
- Αν ωστόσο θέλαμε να τροποποιήσουμε τη λειτουργία της format(), μπορούμε να κληρονομήσουμε τη Formatter και να επαναορίσουμε κάποιες από τις μεθόδους που καλούνται κατά την επεξεργασία της συμβολοσειράς (δεν θα καλυφθεί στο μάθημα αυτό)

ΜΑΘΗΜΑ 8.3: Το module string

2.1. format(): Σύνταξη πεδίων αντικατάστασης

- Η **μέθοδος format()** της string ακολουθεί το συντακτικό:
  - string.format(args)
  - όπου η συμβολοσειρά string περιέχει 0 η περισσότερες εμφανίσεις του {}.
  - Το {} θα αντικατασταθεί με την τιμή του αντίστοιχου ορίσματος (το 1ο {} με το 1ο όρισμα, το 2ο, με το 2ο όρισμα κοκ.)
- Το **{}** καλείται **πεδίο αντικατάστασης** και έχει πλούσιο συντακτικό:

```
replacement_field  ::=  "{" [field_name] ["!" conversion] [":" format_spec] "}"
field_name         ::=  arg_name ("." attribute_name | "[" element_index "]")*
arg_name           ::=  [identifier | digit+]
attribute_name     ::=  identifier
element_index      ::=  digit+ | index_string
index_string       ::=  <any source character except "]"> +
conversion         ::=  "r" | "s" | "a"
format_spec        ::=  <described in the next section>
```

Το πεδίο αντικατάστασης μπορεί να είναι:

- **{}**: 1-1 αντιστοίχιση με θεσιακά ορίσματα
- **{0}, {1}, {2}:** Αντιστοιχεί στις θέσεις των θεσιακών ορισμάτων
- **{kw}:** Αντιστοιχεί σε ορίσματα με λέξεις κλειδια

Θέτοντας στο πεδίο !x (π.χ. {0!x} ή {kw!x} φιλτράρεται το αποτέλεσμα:

- **!s**: Καλείται η str() επί του ορίσματος
- **!r**: Καλείται η repr() επί του ορίσματος
- **!a**: Καλείται η ascii() επί του ορίσματος (built-in που κάνει escape τους χαρακτήρες που δεν είναι ASCII)

Παράδειγμα 3: format_replacement.py

```
print("{}, {}, {}".format(0, 1.1, "str"))
print("{0}, {1}, {2}".format(0, 1.1, "str"))
print("{1}, {2}, {0}".format(0, 1.1, "str"))
print("{arg1}{arg2}".format(arg1=0, arg2=1))
print("{arg2}, {arg1}, {1}, {0}".format(0, 1, arg1=0, arg2=1))

class Time:
    def __init__(self, hour, minute, second):
        ...
    def __str__(self):
        ...
    def __repr__(self):
        ...
t = Time(4, 48, 0)
print("str={0!s}\nrepr={now!r}\nascii={now}".format(t, now=t))
```

- Επιτρέπεται στα πεδία, να τα δούμε ως λίστες ή λεξικά ή αντικείμενα, όπως στο ακόλουθο παράδειγμα:

Παράδειγμα 4: replacement_indexing.py

```
t = Time(4, 48, 0)
print("minutes = {0.minute}".format(t))
l = [1, 2, 3]
print("second = {0[1]}".format(l))
d = {"a": 1, "b": 2}
print("val = {arg[a]}".format(arg=d))
```

Μέχρι στιγμής:

- Έχουμε ορίσει το περιεχόμενο των άγκιστρων ως {xx!y} όπου xx προσδιορίζει τι θα τυπωθεί kai !y αν θα περάσει από το φίλτρο κάποιας dunder method
- Το επεκτείνουμε ως {xx!y:z} όπου z ορίζεται ως:

```
format_spec ::=
              [[fill]align][sign][#][0][width][grouping_option][.precision][type]
fill          ::= <any character>
align         ::= "<" | ">" | "=" | "^"
sign          ::= "+" | "-" | " "
width         ::= digit+
grouping_option ::= "_" | ","
precision     ::= digit+
type          ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" |
                  "s" | "x" | "X" | "%"
```

- **width:** Ακέραιος που καθορίζει το πλάτος εκτύπωσης
- **precision:** Πλήθος σημαντικών ψηφίων
- **align:** Στοίχιση (< αριστ, > δεξιά, ^ μέση, = μετά το πρόσημο)
- **fill:** χαρακτήρας με τον οποίο γεμίζει ο κενός χώρος

Παράδειγμα 5: format_spec.py

```
print("|{:3}|".format(1))
print("|{:6.2}|".format(1.1234))
print("|{:>10.2}|".format(1.1234))
print("|{:_^6.3}|".format(1.1234))
```

- **type:** Καθορίζει τον τύπο δεδομένων που χρησιμοποιείται

type	Επεξήγηση
b	δυαδικός
c	χαρακτήρας
d	δεκαδικός
o	οκταδικός
x, X	16δικός

type	Επεξήγηση
e, E	Επιστ. μορφή
f, F	float
g, G	… docs …
n	number (locale)
%	ποσοστό

- **sign:** εκτυπώνει το πρόσημο
- **grouping_option:** Διαχωριστής χιλιάδων, με , ή _

Παράδειγμα 6: format_spec2.py

```
print("|{:b}|".format(4))
print("|{:x}|".format(1000))
print("|{:<6.2f}|".format(1))
print("|{:>-10_.4f}|".format(5432.12))
print("|{:.2%}|".format(.3015))
print("|{1:_^-100_.2f}|{0}|".format(1000.1234, 2000.2468))

d = {"a": 5, "b": 10.123}
print("|{d[a]:_^-100_.2f}|{d[b]}|".format(d=d))
```

3. Η κλάση Template

- Η **κλάση Template** προσφέρει ένα πιο εύκολο τρόπο για να κάνουμε αντικαταστάσεις σε συμβολοσειρές
  - Ασφαλώς πιο εύκολο από τη format()
  - και με πολύ λιγότερες δυνατότητες
- Κατασκευαστής:

Κατασκευαστής	Επεξήγηση
Template(string)	Παίρνει ως όρισμα μια συμβολοσειρά που περιέχει placeholders (θέσεις που θα αντικατασταθούν από μία άλλη συμβολοσειρά)

  - Ένας placeholder αρχίζει με το $ και ακολουθεί ένα λεκτικό (identifier) που θα χρησιμοποιοιηθεί για να κάνουμε την αντικατάσταση.

Παράδειγμα 7: template_init.py

```
template1 = Template("$name lives in $place")
template2 = Template("$name makes $salary $$")
```

- Χρησιμοποιούμε τις εξής μεθόδους για να κάνουμε την αντικατάσταση:

Μέθοδος	Επεξήγηση
substitute(d)	Κάνει τις αντικαταστάσεις χρησιμοποιώντας λεξικό με κλειδιά τους placeholder και τιμή τη συμβολοσειρά αντικατάστασης.
substitute(**kwds)	Η αντικατάσταση γίνεται με βάση τα ορίσματα με λέξεις-κλειδιά. Κλειδί=placeholder, Τιμή=συμβ/ρά

- Είναι εφικτός και ο συνδυασμός των δύο ως:
  - substitute(d, /, **kwds)

Παράδειγμα 8: template_substitute.py

```
from string import Template

template1 = Template("$name lives in $place")

sub11 = template1.substitute({"name": "Bob", "place": "Athens"})
print(sub11)
sub12 = template1.substitute(name="Bob", place="Athens")
print(sub12)
sub13 = template1.substitute({"name": "Bob"}, place="Athens")
print(sub13)
```

- Επίσης προσφέρεται η μέθοδος:

Μέθοδος	Επεξήγηση
safe_substitute (d, /, **kwds)	Αν κάποιος placeholder λείπει από το d ή το kwds, δεν προκαλείται KeyError αλλά το placeholder θα εμφανίζεται ως έχει στην αντικατάσταση.

Παράδειγμα 6: template_safe_substitute.py

```
template2 = Template("$name makes $salary$$") # $$ escapes $
sub21 = template2.safe_substitute({"name": "Bob", "salary": 1000})
print(sub21)
sub23 = template2.safe_substitute({"name": "Bob"})
print(sub23)
```